RasPi

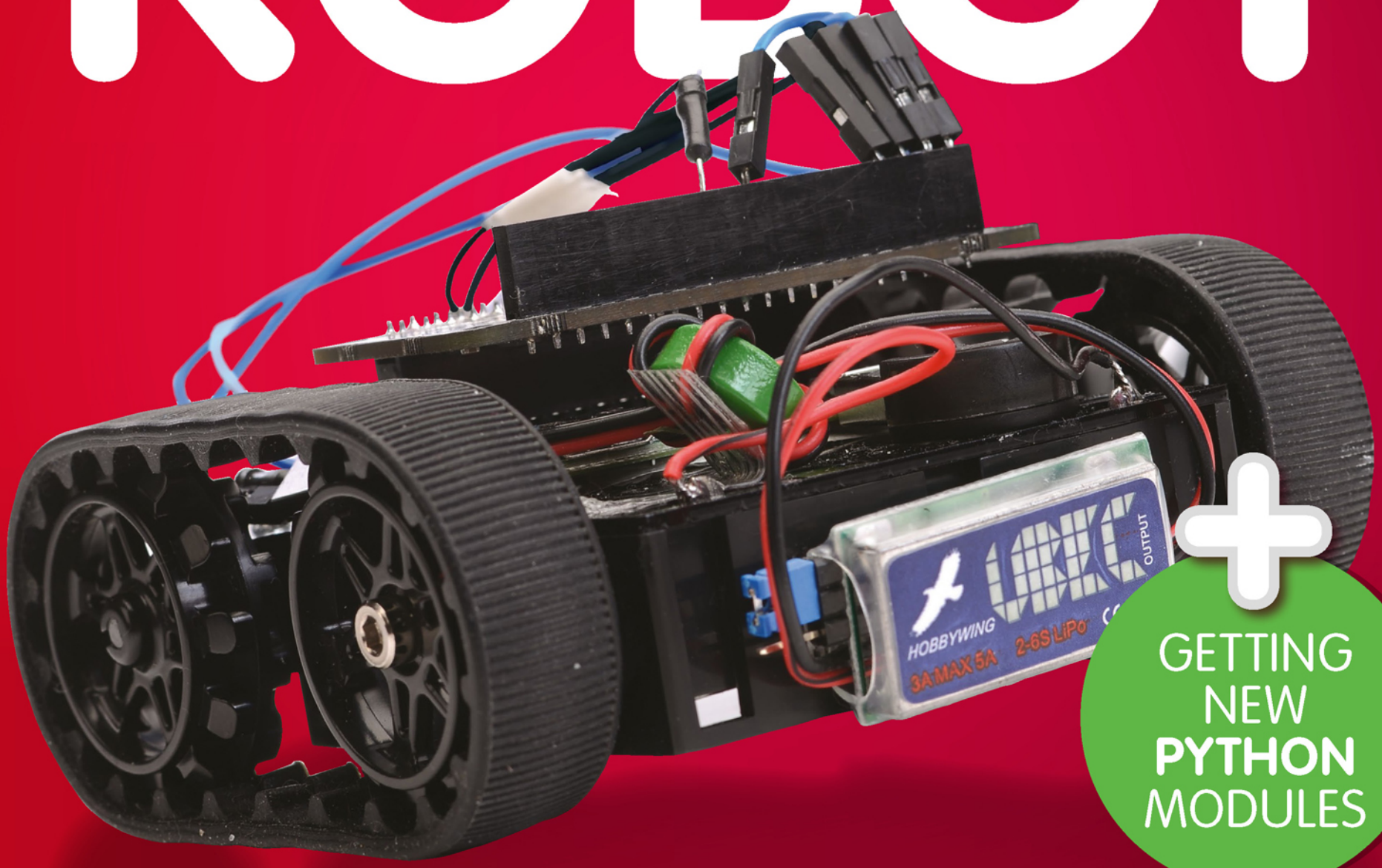
DESIGN
BUILD
CODE

33

Get hands-on with your Raspberry Pi

BUILD YOUR OWN

ROBOT



+ GETTING NEW **PYTHON** MODULES

Plus Make a Pi-powered baby monitor

Welcome

With **Robot Wars** back on our screens, why not try building your own Pi-powered android? In this issue of RasPi we'll show how to make and program a robot with a Raspberry Pi Zero, which you can then control with a gamepad. If you want to go one step further, you can also make it autonomous, so that it can dodge obstacles by itself. However, we make no guarantees that it would be able to avoid Sir Killalot in the arena for long! Also this issue, you can find out how to turn a Raspberry Pi into a router and a baby monitor, as well as combine it with code, and get involved with Berryconda to leave Raspbian Python modules behind. We also find out how the Pi is being used to solve the water crisis.

April

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@futurenet.com

Contents

Build an explorer pHAT robot

Make your own autonomous Raspberry Pi-powered robot using a Pi Zero, Zumo chassis and Pimoroni Explorer pHAT

Here is everything you need to know to make your own Raspberry Pi robot. To start, we break down a robot into all its component parts and give you an insider's perspective on batteries, motors, input controllers and more.

01 What exactly is a 'robot'?

A robot is considered to be any machine that can perform a task or series of tasks, either autonomously once programmed or with direct instruction. Robots come in many shapes and sizes – they can be used for manufacturing, for search and rescue, and for just having fun and seeing what you can create. The hobbyist creations you see at events like Pi Wars are wheeled or tracked, disconnected from any wires, and capable of being driven with a gamepad. We're not going to be too strict about what constitutes a robot; we'd like to leave that up to you.

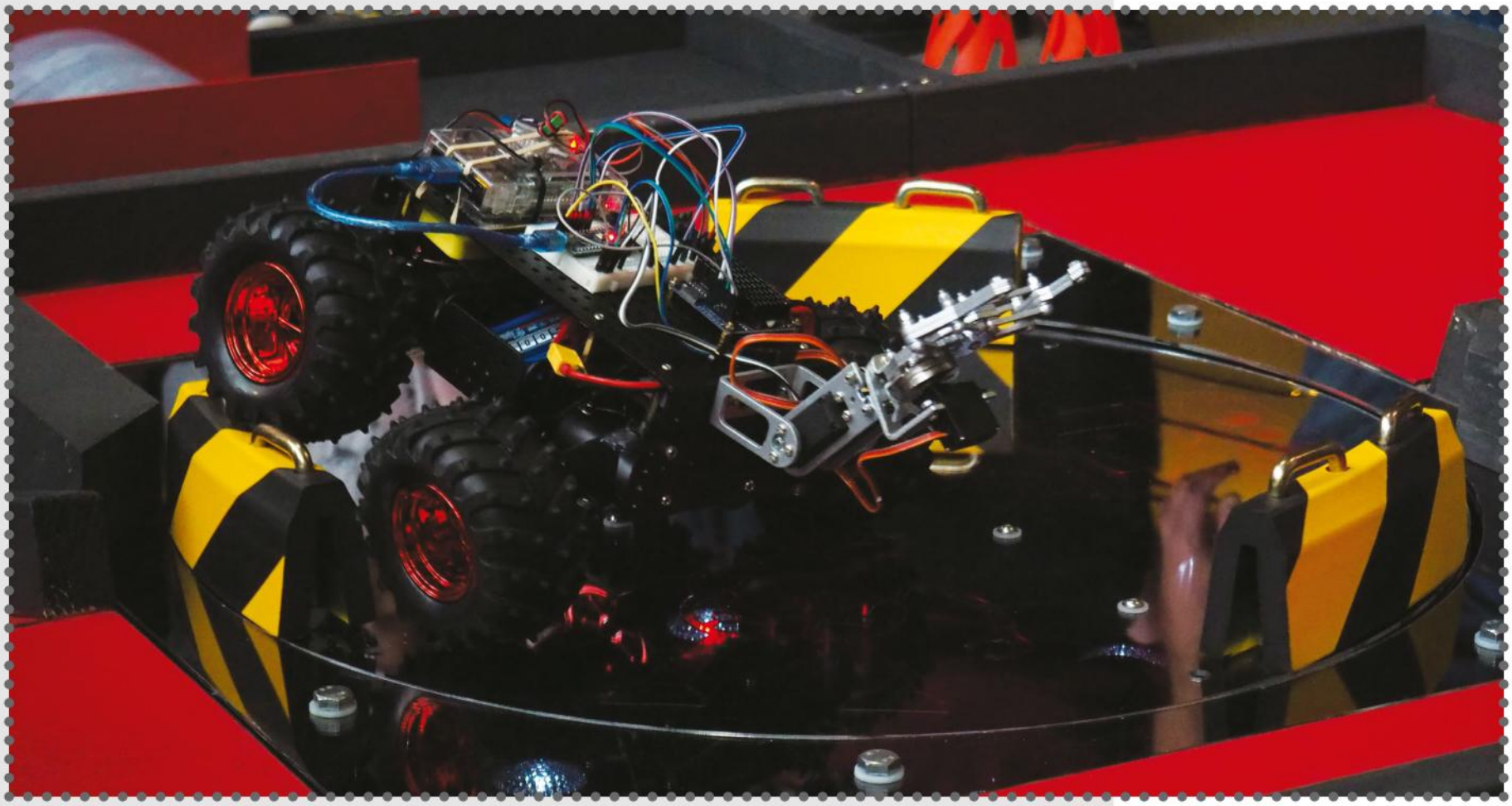
02 The chassis

The chassis is at the heart of the robot, and its size and shape dictates everything else that you add to it. Choose carefully between big or small, strong or light, heavy or nimble. If you are going autonomous then you will need a base with plenty of space for expansion. At Pi Wars we saw RC cars re-purposed with brand new electronics; this can be a cheap way of getting a great finish. High-end kits such as the Dagu Wild Thumper are made from steel and have enough power to pull a chair across a room. DIY platforms are the most flexible and rewarding, though; we saw laser-cut wood, acrylic and even 3D-printed parts from the Hitchin Hackspace team.

THE PROJECT ESSENTIALS

We sourced most of these components from Pimoroni for around £35. We assume you already have your Raspberry Pi Zero, microSD card and few AA batteries in the back of a drawer.

- **£15.00 Polulu Zumo chassis**
- **£10.00 Explorer pHAT**
- **£10.00 2x Metal-gear motors**
- **£2.00 Battery UBEC (eBay)**



Above With enough time and dedication, your robot could be re-purposed so it's strong enough to pull a chair across a room

03 Power source

It is best to run your motors and control circuits from two separate power sources. This helps isolate your Pi, motor controllers and sensors from noise and potential surges. For the experienced user, LiPo batteries are recommended, but we must warn you that all safety notices must be read and followed thoroughly before charging or handling them. Safer alternatives exist in USB power banks, AA batteries and NiCD 9V batteries.

04 Get your motor running

Brushless motors for RC cars typically run at high speeds and can make your robot great at travelling in a straight line, but hard to control over obstacle courses and for autonomous navigation.

Brushed motors can be driven with a motor driver, but you must refer to the stall current of the motor when choosing the board.

Motor controllers are driven with a PWM (Pulse Width Modulation) signal generated by the Pi or an add-on board. It's a good idea to buy at least one or two spare motors for each project – and remember to be kind to them because they can break when misused. The Zumo chassis uses micro metal-geared motors from Pimoroni.

05 The motor controller

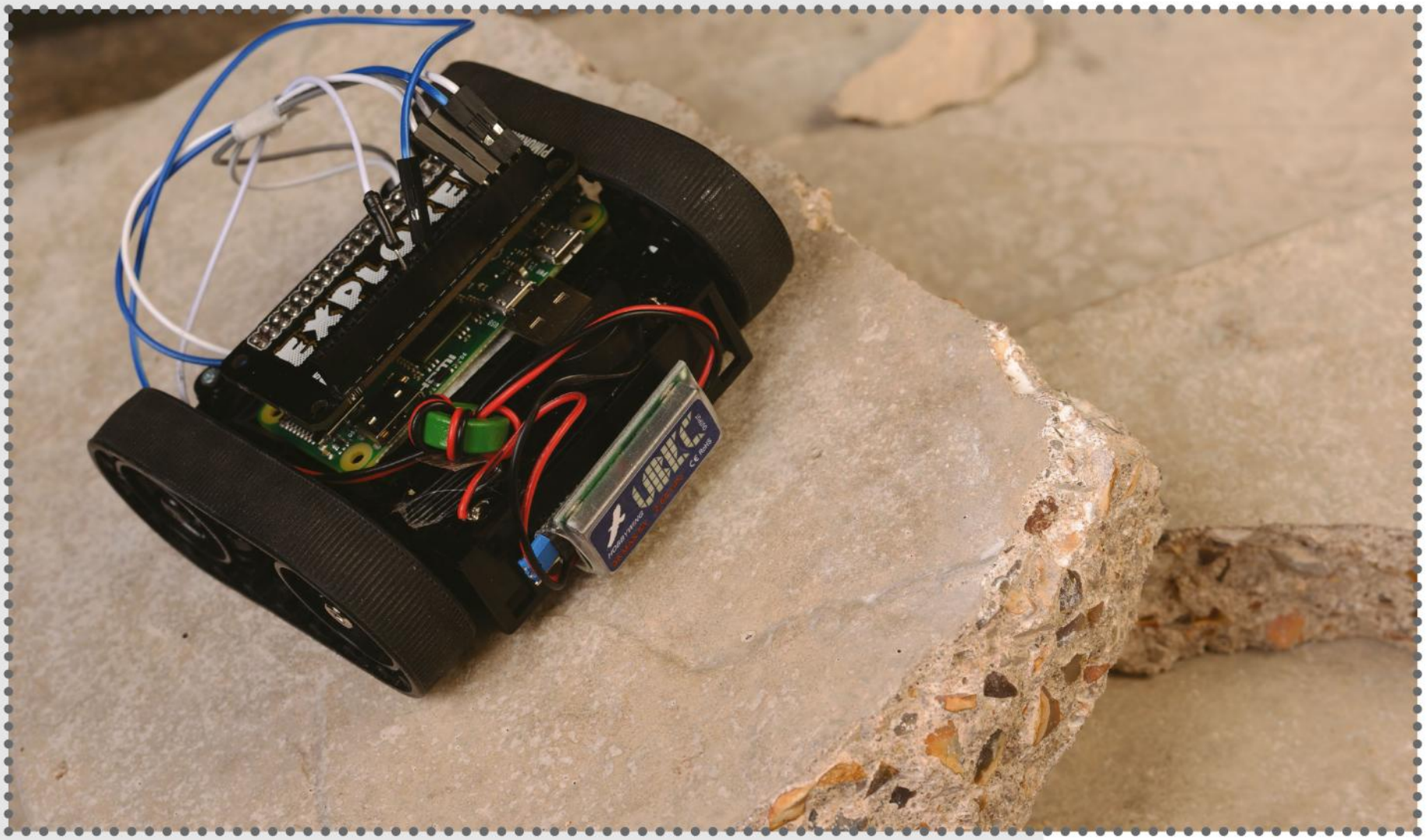
The type of motor and its peak current draw will dictate your choice of motor controller. For small motors, a cheap chip like the L298N (£2) may well suit your needs. PiBorg produces a much more efficient board which can drive large currents, but the cost will increase up to around £30.

06 Understanding gear ratios

Gears are expressed as the ratio of input speed to output speed. For instance, 35:1 means the motor shaft (driver) turns 35 times as fast as the driven shaft (where the wheel is mounted). The higher the number, the slower the driven shaft will turn, but with a higher torque (twisting force).

With the Dagu Wild Thumper 4WD robot (available at **robosavvy.com**) a ratio of 34:1 gives a top speed of 4.5mph with a stall current of 5Kg.cm; in contrast, a ratio of 75:1 has a slower top speed of 2mph but has a much higher torque at 11Kg.cm.

For more detailed infomation on gears, check out this great slide deck from Bowles Physics: **http://bowlesphysics.com/images/Robotics_-_Gears_and_Gear_Ratios.pdf**



Above Robots can be capable of scaling all different kinds of terrain

07 Controller/input options

Most robots will need manual input through a joystick or gamepad. We've had good success with genuine Wiimotes and PS3 controllers which use Bluetooth. The Xbox 360 controller also works well but adds a large dongle to the Pi and draws more current than a Bluetooth adapter. Separate your code into different classes so that, if you need to, you can swap controllers in the future.

08 LED outputs

When starting and stopping your robot, or just turning the motor speed up or down, it's important to have visual feedback. The simplest way to achieve this is through LEDs plugged directly into the GPIO headers of your Pi with an appropriately-rated resistor. For our first robot, a flashing sequence was performed when

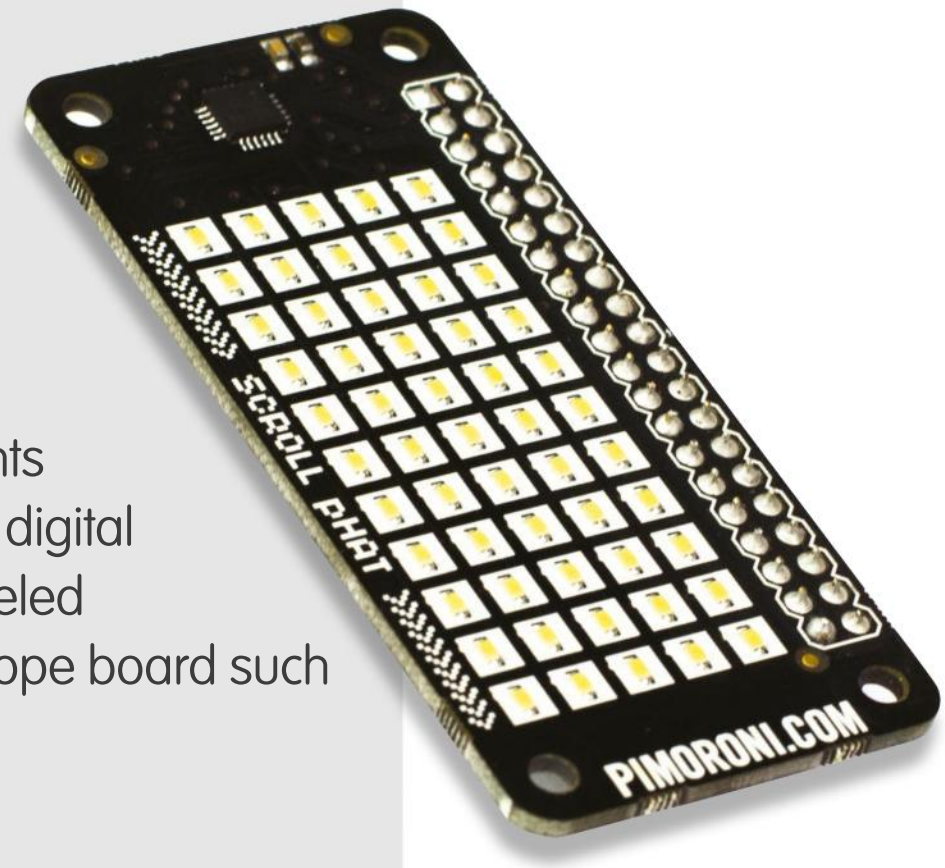
the Bluetooth service had started. We added separate LEDs for when the movement speed was turned up and down. If you are creative or just have a lot of space on the robot's top level, you can attach an LCD display at the same time as everything else for communicating data from the sensors.

09 Starting out with sensors

There are many types of sensors available and each has its own protocol or wiring scheme. We would suggest keeping to well-known sensors that are easy to connect. The Parallax ultra-sonic Ping sensor gives the distance to the nearest object in centimetres, helping you avoid crashing into walls. A pair of infrared line sensors can be used to follow a dark line – you could use black insulation tape on the kitchen floor.

10 Cameras, GPS, compasses and gyros

A camera can be added to the Raspberry Pi Zero v1.3 for around £4 for the cable and £10-22 for a camera. This means that as you are navigating around the kitchen or the garden you can take pictures or stream live video back to your control station (laptop). For something more advanced, a Raspberry Pi robot can be made to follow a series of GPS waypoints with a USB or serial GPS module and a digital compass. You can even build two-wheeled balancing robots by using a tiny gyroscope board such as the 9-DOF from Adafruit.



11 Autonomous control

There are two main types of autonomous control programs for a robot: open and closed loop. In an open loop, a robot will power its motors at a constant speed regardless of the environment, meaning a small turn on carpet will translate to a 360-degree manoeuvre on tile flooring. Generally, an open loop is easier to start with because it has absolutely no feedback involved. More advanced robots need to use feedback from sensors and inputs to adjust their speed or movements. An example is a robot on the flat versus one going uphill; a closed loop involves a wheel encoder so that instead of travelling at 50 per cent voltage we can task the robot with moving at a minimum of 100rpm.

Build an explorer pHAT robot

Build your explorer robot, soldering wires, connecting the tracks and setting up the Pi

In part one, we read Wikipedia's definition of a robot as a machine capable of carrying out a series of actions automatically (paraphrased). We then gave an overview of all the considerations you may have for your build, from the chassis to the motors to the sensors you may want to add for autonomous tasks. In this section, we invite you to join us as we build our own model robot. If you follow along, you should have everything to get you started, especially if this is new territory for you.

We are going to build your explorer robot from scratch running through each step needed, including soldering and putting all the pieces in place. We start with looking at the full parts list, the chassis kit and the other non-robotic parts. We will solder header pins onto our explorer pHAT from Pimoroni, flash our SD card and run the initial setup to enable the pHAT.

THE PROJECT ESSENTIALS

Article's Github repository (https://github.com/alexellis/zumopi)

Raspberry Pi Zero

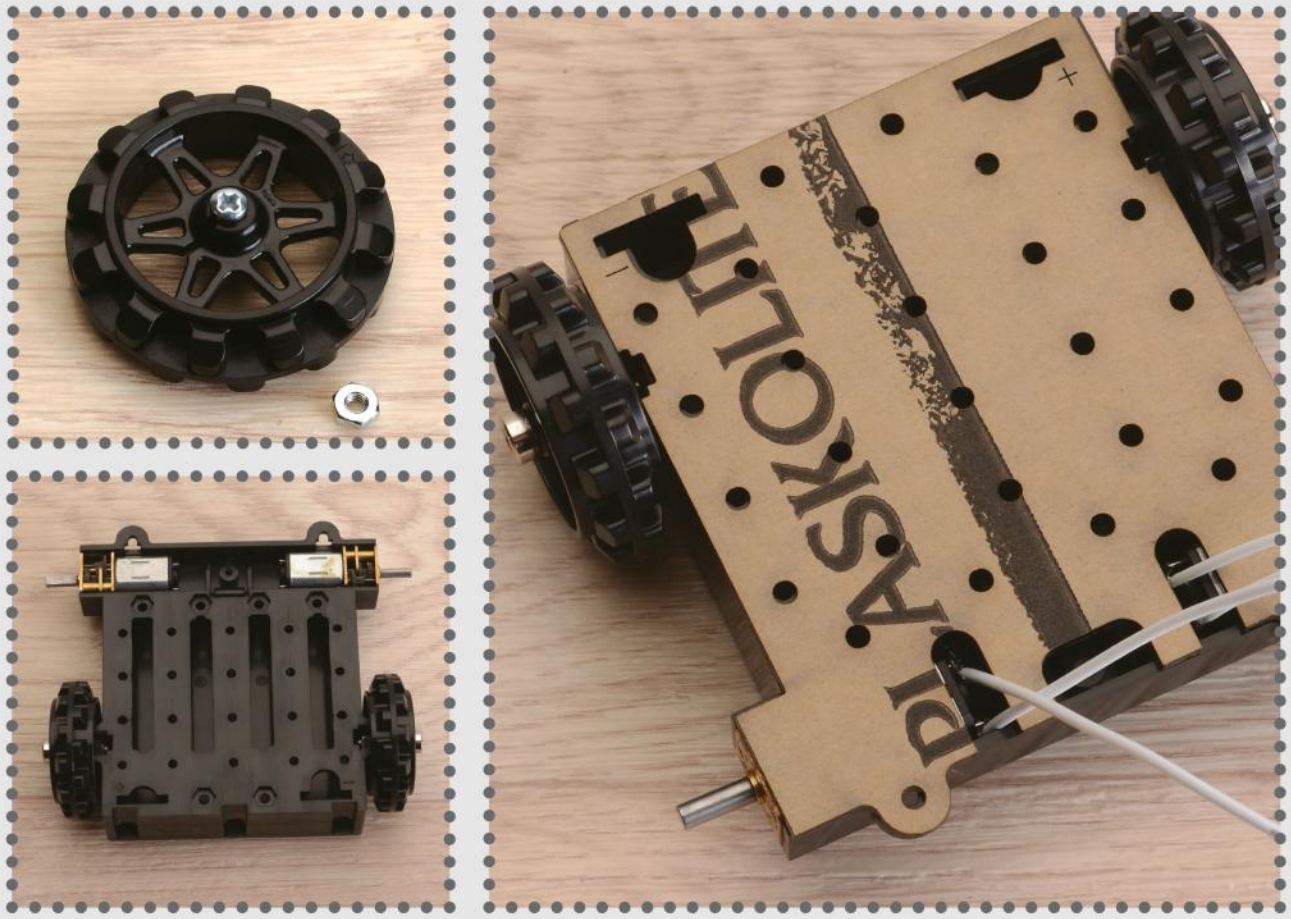
microSD Card

All other kit is listed in step 1

01 The full parts list

In addition to the Pi Zero you will need:

- A Pimoroni explorer pHAT



- A Polulu Zumo Chassis kit (available at Pimoroni)
- 2x micro-metal gear motors (available at Pimoroni)
- Cross-head screw driver
- Soldering iron, flux & solder
- 3A 5-6v UBEC (from HobbyKing or eBay)

If you are struggling to find a Pi Zero head over to http://stockalert.alexellis.io/ for a live stock count at three major outlets.

02 The chassis kit

The chassis is made up of three main parts – the largest piece is where we attach the wheels and tracks,

but also holds the 4xAA batteries. It has a battery lid that, if lost or damaged, will mean the batteries may drop out, so take good care of it. There is an acrylic cut-out that mounts on the top with bolts – peel off the brown protection paper.

03 The axles and free-wheels

The robot has two free-running wheels and two that are directly driven by the geared motors.
Insert the two metal hubs into the plastic wheels then attach to the chassis, tightening up the bolt on either side. You may find that a small Allen or hex key helps here, but do not over-tighten the bolt.
Both wheels should spin freely, and now we can move onto the motors.

04 Preparing the motors

The motors currently come with two small solder pads – one for positive and one for negative. We suggest clipping

the ends of two sets of white and black or red and black jumper wires leaving one male dupont (jumper-style) end on each wire. Next carefully strip 2-3mm of wire from the bare end and solder this to the pads on the motors. A little flux will really help and cut off any excess wire afterwards.

05 Installing the motors

Once the solder joints have cooled down, turn off the solder iron and place the motors with the shafts in the small slots at the opposite end of the chassis. They have a friction fit that will be made tighter when the top plate is screwed down. You can temporarily place the top plate over the motors to help with deciding where to route your wires. We used the two grooves to the left and right.

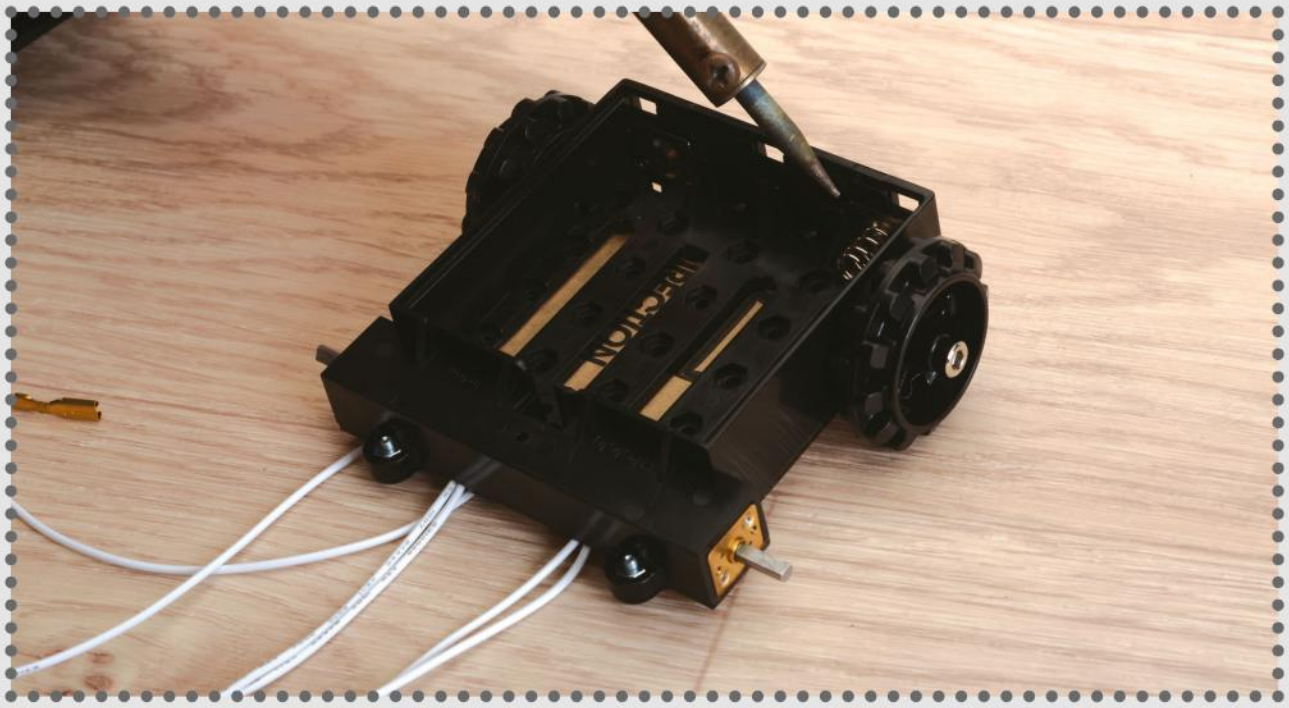
06 Secure the top plate

Securing the top plate can be a bit fiddly. Place the robot chassis with the battery compartment pointing up. Find the first two bolts and nuts and

then drop a nut into the slot by the motors and turn it until it sets. Use a small crosshead screwdriver to tighten it.

07 Solder the battery contacts

Now let's look at the battery contacts. They come in five separate parts and when we first tried to put these together, we needed a lot of trial and error. One thing you can do up front is solder the positive and negative terminal. Go ahead and solder a small piece of jumper wire through the positive and negative terminal; these are identified through a tiny loop.



08 Inserting the battery contacts

We found that the positive and negative terminals could move around, especially when no batteries were inserted. You may want to put a dollop of hot glue or hockey-stick tape behind the positive and negative terminals so that they remain in place when going over bumps or when inserting new batteries. You will find that the wires thread through the bottom of the compartment and we now have six sets of wires in total.

09 Mounting the front wheels

We will now mount the front set of wheels that drive the

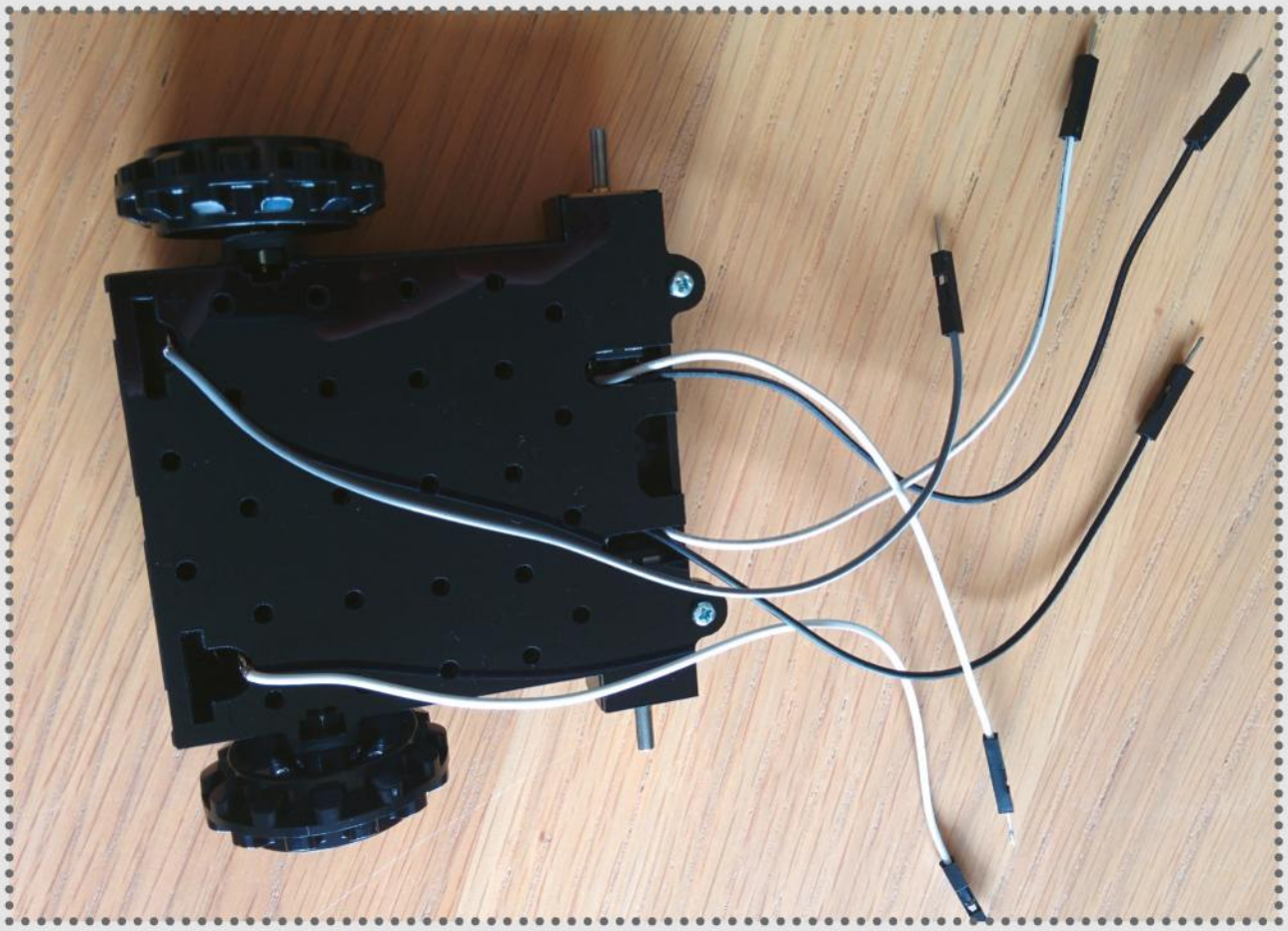
tracks. Find the shallow side of the wheel and let it point out, then slowly but firmly push the plastic wheels down onto the shaft of the motor. Try to make sure these line up with the free-running wheels we installed in step three. You will be able to turn the motors by hand but it is not recommended – just check that they are able to turn freely without rubbing on the plastic wheel arches.

10 The axles and free-wheels

Our robot runs on two sets of rubber tracks, which are elastic enough to stretch over the two sets of front and rear wheels. With the teeth of the track facing in, place the track on one wheel then stretch it very carefully over the other side. Before going any further, find two more nuts and bolts and secure the opposite end of the chassis next to the battery terminals. You will need your crosshead screwdriver again.

Zero-sized motor controllers

The explorer pHAT is one of our favourite motor controllers for the Pi Zero. Having analog inputs and ouputs along with the motor control means it can be used for many different purposes. There are other motor controller boards designed for the Zero by 4tronix, PiHut and PiBorg. PiBorg's ZeroBorg was a KickStarter project that has shipped to backers.



11 Check and tighten everything

We have now installed the drive motors, both sets of wheels and tracks. We took the cover off the acrylic top plate and secured it with two sets of nuts and bolts. We have installed the battery contacts and cover. There are three pairs of wires: two sets for the left and right motors and two for the batteries. Install four AA batteries and touch together the battery wires with each of the motors, you should see each side turn. If not then go back and check all the solder joints.

12 Preparing the explorer pHAT and zero

If you have a Pi Zero without a header attached, then go ahead and solder a 40-pin male header, making sure to use as little solder and flux as possible. Next mount the 40-pin female header that came with your explorer pHAT and solder each joint – we find that a peg or a piece of Blu-Tack can help make sure the header is attached squarely without pointing off at an angle. Now solder the single-row female pin header on the opposite side.

13 Adding the UBEC

The UBEC is a battery elimination circuit, it will make sure that 5v is supplied to the Raspberry Pi. The component comes with a female header for the output, you can plug this straight into the +ve and -ve connectors on the explorer pHAT. The input side is two wires that will need

Smart Soldering

Good solder joints use the smallest amount of material possible to make a clean joint. A flux pen can be a good way to help solder two wires or metal surfaces. If you add too much solder then a 'solder sucker' or desoldering braid can be used to remove excess material. We would also highly recommend you purchase a fire retardant mat.

FROM FRESH
NEWSPAPERS TO
ALL NEW MUSIC RELEASES

ONLY ON

AVXHOME.IN

to be attached to the battery contacts with solder, twisting or non-conductive tape.

14 Connect up the motors

At this point our motors can be plugged into the explorer pHAT using the motor 1 and motor 2 connections on the front of the explorer pHAT. You may find that some of your wires are too long; there should be no harm in cutting them down to the size and re-soldering them. We think that the best way to do this is through trial and error.

15 Fire it up and set up Python

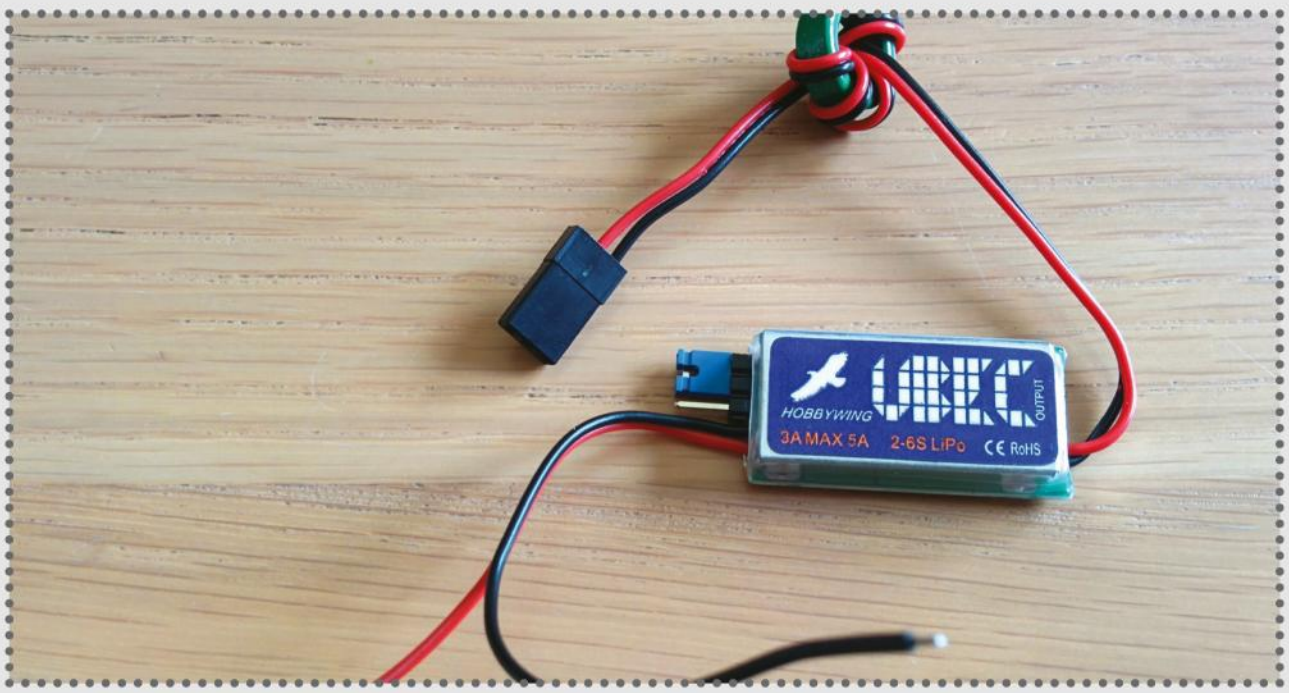
Our robot will be programmed in Python, so flash Raspbian or Raspbian Lite to your SD card and boot up the Pi. Providing everything starts up as it's supposed to, configure the internet and then run Pimoroni's installation script for the explorer pHAT Python library.

```
```

```
$ curl -sS get.
pimoroni.com/explorerhat
| bash
```

Now check that the library can control the motors by running this Python code:

```
import explorerhat
import time

explorerhat.motor.one.
forwards()
explorerhat.motor.two.
forwards()
time.sleep(2)
explorerhat.motor.one.
stop()
explorerhat.motor.two.
stop()
```



Control your creation

Give your explorer robot its legs with our robot-coding tutorial

THE PROJECT ESSENTIALS

Article's Github repository (https://github.com/alexellis/zumopi)

Explorer Robot as built in part two

Genuine Wiimote (optional)

This tutorial builds the software to control your explorer robot so you can let it loose in the living room, the office or even in the garden. We will put together a program to drive the robot from a console, letting you type in WASD as if in a computer game. This gives you a chance to get comfortable with how the robot moves and how the code is put together. We will then go on to integrate a Nintendo Wiimote controller, which makes for a much more natural controller than a keyboard.

The code will be written in Python, which is pre-installed on Raspbian, and wherever possible will be compatible

with both Python 2 and 3. The entire code will be open source, so if you want to suggest an enhancement or to correct a bug, you can submit a pull request or raise an issue on Github.

01 Going off-grid

Our robot chassis was originally intended to be run by an Arduino with a much lower current consumption than the Pi Zero. While a range between 5-36V can power some Arduino boards, our Pi needs a clean 5V from the UBEC. Alkaline batteries when brand new and unused have a voltage of around 1.65-1.7v, meaning that the input to the UBEC will be around 6-7v. Some of that will be wasted during the conversion to 5v and the rest will drive the motors and the Raspberry Pi at the same time.

02 Time for a pit-stop

While batteries are needed to go wireless and to roam around we suggest you set up a mini-workshop for your explorer robot that you can

bring it into to download new code and debug problems when things aren't going to plan. Here you will have access to a USB power supply, a powered USB hub, a HDMI TV or monitor, a Bluetooth dongle, Wi-Fi dongle and a keyboard. Make absolutely sure that you disconnect your batteries before plugging in.

03 Calibrate your motors

We will now calibrate the motor's direction so that it matches up with the software. Dock your robot and put it on an empty matchbox so that its tracks can run free without taking the leads with it. Now use Python to check that each motor moves the tracks forward when invoked from code. Either type the code into an interactive Python prompt or use the file **`motor_tests/forwards.py`**.

04 Turn left and right

In order to turn left or right we will simply ask one motor

to move forwards and the other to move backwards. The motors you need to turn will depend on how you plugged in motors one and two.

We have designated motor one as the left and motor two as the right. Therefore to test turning, let's try this to turn right.

```
import explorerhat
import time

explorerhat.motor.one.forwards()
explorerhat.motor.two.backwards()
time.sleep(1)
explorerhat.motor.one.stop()
explorerhat.motor.two.stop()
```

This file also exists as **motor_tests/turn_right.py**.

To turn left, you do the opposite action, meaning motor one goes backwards and motor two goes forwards.

05 Work with abstractions

If you look over the code so far there is a lot of repetition, so let's create an abstraction where we can hide away the details of how the motors work and which motors need to go backwards or forwards in order to move.

To try out the Motors class, type in the following or use **motor_test/motors_v1_test.py**.

```
```
import time
from motors_v1 import Motors

motors = Motors()
motors.forwards()
time.sleep(1)
motors.stop()
time.sleep(1)
motors.backwards()
time.sleep(1)
motors.stop()
```
```

```
```
import time
import explorerhat
```

## Rechargeable vs alkaline batteries

For our UBEC to function correctly, we need our four AA batteries to provide over 5V collectively. Rechargeable AA cells generally give a nominal 1.2v when fully charged, totalling 4.8v, which is much less than we need. Alkaline batteries often start at a nominal value of 1.6-1.7v, meaning we will have over 6V and the UBEC can run properly. An alternative to using AA batteries is to strap a USB power bank to the flat side of the robot with cable ties.
```

```
 class Motors:
  def forwards(self):
   explorerhat.motor.one.
forwards()
   explorerhat.motor.two.
forwards()
  def stop(self):
   explorerhat.motor.one.
stop()
   explorerhat.motor.two.
stop()
  def backwards(self):
   explorerhat.motor.one.
backwards()
   explorerhat.motor.two.
backwards()
  def left(self):
   explorerhat.motor.one.
backwards()
   explorerhat.motor.two.
forwards()
  def left(self):
   explorerhat.motor.one.
forwards()
   explorerhat.motor.two.
backwards()
 ```
```

## 06 Create a REPL

Now we have a motors class, we can make our own **REPL** (read-evaluate-print
```

loop) – this is similar to what you get if you type in Python without specifying a file.

First, we read input from the terminal. Then we translate that to a motor direction and move the robot, then we print back to the user what we understood and start over again.

Python provides us with a convenient method for this called raw_input(). We can call it in a loop and then use: **sys.stdout.write** instead of print so that the cursor stays on the same line. We ignore all input other than 'q', which we use as our signal to exit the program:

```
import sys

last = None
while(last != 'q'):
  sys.stdout.
write("Command: ")
  sys.stdout.flush()
  last = raw_input()
  print("You said: " +
last)
```

07 Combine the REPL and Motors class

Let's now start to interpret what we read into the inputs for the motors. Now you can choose whether to let the robot move until a new command is entered, or for a short period and then immediately stop. We prefer the second option because it is easier to control. **motor_repl_v1.py**

08 Abstract the REPL

The **REPL** can be enhanced in many ways, including allowing the move/turn time variables to be edited. Let's improve the structure of the code by introducing some more classes. Classes do one thing really well and help us swap out functionality later on, and even write automated tests. This step would allow us to re-use the same program later on with a gamepad or Twitter feed. You can check out the enhancements in **motor_tests/motor_repl_v2.py**

09 Explanation of the classes

We have introduced three types of classes: one class to read terminal input and convert that to a Command, several classes to represent each command such as Forwards/Left/Right and Quit and then another class called Robot that will run the whole example.

Our new loop is cleaner to read and allows us to update only one thing at a time, meaning changes are less likely to create unexpected problems. We have also named our code so that it is clear what its job is; that means we can save on messy or confusing comments.

10 Repurpose a Bluetooth gamepad

In this step we begin introducing a gamepad – the Nintendo Wiimote. You may have one gathering dust or packed away in the attic, but if

you decide to buy one online make sure that it is a genuine original item, otherwise it will not work with the official Python library. It can also be difficult to find the perfect match in a Bluetooth dongle for your gamepad. We found that a Genuine PS3 controller would not operate with a £1 dongle, but with a premium dongle worked fine. We tried reproduction PS3 gamepads and Wiimotes but we couldn't get either to work.

11 Configure Bluetooth

Raspbian Jessie comes pre-loaded with the requisite system utilities to access Bluetooth accessories. If you are using a different system then look for **Bluez tools**. We will be using the cwiid library to provide our Wiimote interaction. Bear in mind that there are other libaries available and you may want to try them out too. Now install cwiid with **apt-get:**

Upgrading your controller

The Wiimote is an easy controller to work with, but it has a limited amount of buttons. If you want more options, then we suggest getting a genuine PS3 controller. This gives you many more buttons and two separate axes, which can be used to steer the robot, switch between modes or command the robot in other ways. An example would be assigning the select button to take a photo and upload it to Twitter.

```
sudo apt-get install
python-cwiid
```

The following code (also in **wiimote_tests/pair_v1.py**) will pair to the gamepad and then vibrate or 'rumble' it for half a second. Before running the code, hold down A and B or the button under the battery cover to enter pairing mode.

```
import cwiid, time
wii = cwiid.Wiimote()
wii.rumble = True
time.sleep(0.5)
wii.rumble = False
```

12 Read the buttons

Reading buttons is not event-driven; it's something that we need to do in a loop using polling (continual checking). After pairing in the line **`wii = cwiid.Wiimote()`** we start integrating a button's mapping. The code below

in **wiimote_tests/buttons_v1.py** will print out on the terminal when you press either A or B. Notice that the message is repeated for as long as the buttons are pressed – we've added a pause to stop us from overloading the Pi. When finished, hit **Ctrl + C** to quit.

```
import cwiid, time
wii = cwiid.Wiimote()
wii.rpt_mode = cwiid.RPT_BTN
while(True):
  buttons = wii.state['buttons']
  if buttons & cwiid.BTN_A:
    print("A pressed, go forwards")
  if buttons & cwiid.BTN_B:
    print("B
```

```
pressed, go backwards")
  time.sleep(0.02)
  ```
```

## 13 Create a WiimoteReader class

Now that we can read inputs from our gamepad, let's create something to convert that into commands that our robot can understand. Since we will be doing this in its own class, we can then test the code without needing to move the robot. Below is the code from **drive_v1/wiimote_test.py**, which will print out the command as we press buttons on the gamepad. Use **Ctrl + C** to stop the example.

```
  ```
  import time
  from wiimotereader
import WiimoteReader

  reader =
WiimoteReader()
  cmd = None
  while(True):
    cmd = reader.read()
```

```
    print(cmd)
    time.sleep(0.25)
```
```

## 14 Finish the control software

We can now control the robot through individual command classes and have them created either by a terminal and keyboard or by a Wiimote gamepad. If you were to download the Python library for a PS3 controller all you would need to do is to convert the inputs into the command classes that we are already working with.

Taking the example we had in Step 13, let's go one step further and start executing the commands as they are interpreted by the **wiimotereader.py** code.

## 15 We could stop here

We could stop the tutorial at this point because we have fully functioning manual control of our robot with a gamepad, but there are still a
```

few important steps we want to take you through. We need to make our program auto-start so that when it's powered by batteries and our sole USB port is populated with a Bluetooth dongle, we can still load the control software. We can create a simple startup task with Cron, a tried and tested Unix scheduler. Add an @reboot directive to your crontab file. Type in **`chmod +x start.sh`** (hit enter) then **`crontab -e`:**

```
@reboot /home/pi/explorerrobot/part3/drive_v1/start.sh
```

Systems could also be used to create a configuration and start-up file; it has many advantages but those layers also add complexity.

16 Run other commands

In addition to auto-starting, we need a safe shutdown

Should you code on Pi directly?

With modern, open-source code editors available such as Visual Studio code (**https://code.visualstudio.com/**) and Atom (**https://atom.io**) there is every reason to write the majority of code on your laptop or desktop PC before using scp/sftp or git push/pull to transfer the code across to the device. We use a mixture of coding on the device and then consolidating that into our Github repository or folder on a PC or laptop. This also means you always have a second copy of your work if the Pi's SD card corrupts.

mechanism, so that the Pi will perform a clean power-down leaving our SD card in good condition. Linux's halt command is a good candidate for a safe shutdown. Import Python's os module and use it to execute a command, here's an example with uptime:

```
>>> import os
>>> os.system('uptime')
 21:34:11 up 16 min,  1 user,  load average: 0.00, 0.03, 0.07
```

All we need to do is to add a shutdown class and then have it parsed by the WiimoteReader class. We've picked the Home button for shutdown because it's hard to press by accident.

17 Implement a Shutdown class

The shutdown class should look like this:

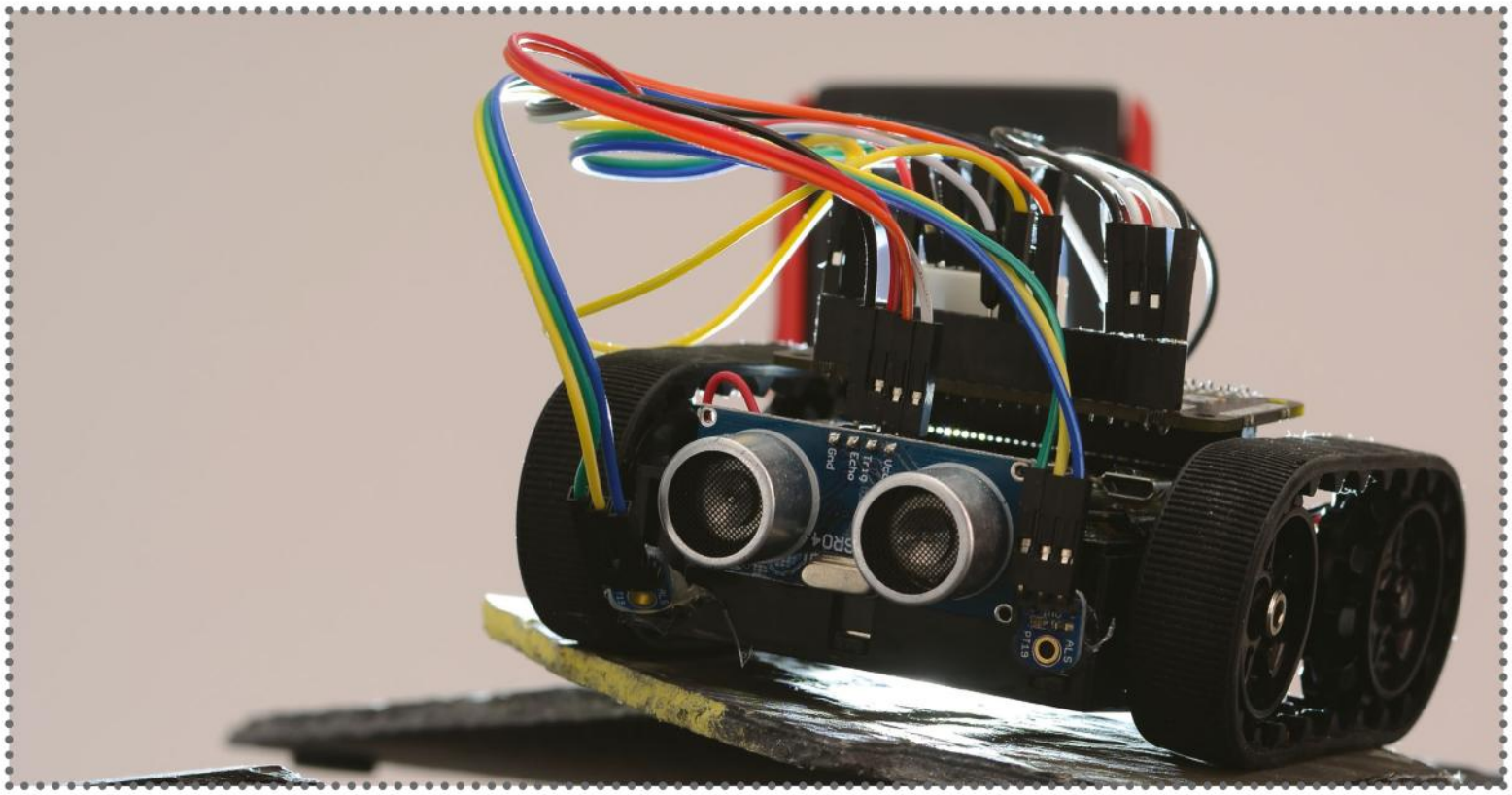
```
```

```
import os

class Shutdown:
  def Execute(self,
motors, move_delay,
turn_delay):
    os.system(“sudo
halt”)
    return False
```

The updated parsing code can be found in WiimoteReader. You are now ready to unplug the USB hub and PSU – replace the batteries and plug in 5V/GND from the UBEC. This will immediately start up the Pi. Start holding the pairing button down when the Bluetooth dongle starts to flash.

Add autonomous features

Give your explorer robot its legs with our robot-coding tutorial



Take your robot to the next level and add self-driving capabilities through sensors and a feedback loop. We start off by looking at open versus closed loops for control, then create a simple obstacle-avoiding behaviour.

Once our robot can avoid walls, we will add a second behaviour to follow a light source such as a torch being shone at the left or right side of the robot. The programs are only a starting point for you to adapt and tailor to your needs. The light-following program could be adapted into a line-following behaviour with the use of two infrared LEDs.

THE PROJECT ESSENTIALS

Article's Github repository (https://github.com/alexellis/zumopi)

Explorer Robot as built in part two

Wi-Fi dongle

Ultrasonic sensor

2x analog light sensors

The examples will be written in Python, which is pre-installed on Raspbian. Wherever possible, the examples will be compatible with both Python 2 and 3. All the code will be open source, so if you want to suggest an enhancement or to correct a bug you can submit a pull request or raise an issue on Github. Please also check Github for any errata or any changes we may make to keep the article up to date.

01 There is no failure, only feedback

As we begin to design an autonomous behaviour for our robot, we will see that navigating even a simple, predictable environment such as a kitchen floor can be a very tricky task. Take our robot from part three of the series as an example. If we drive around on a thick carpet, grass, concrete or lino flooring, then you will notice big differences in moving and turning speed. If we turn our robot at 100 per cent motors for 0.5 seconds on lino it may perform a 360, whereas on thick grass it may barely move and on carpet it may turn exactly 90 degrees.

02 Are you open or closed to feedback?

During manual control we

observe the robot's driving performance and react by correcting it on the gamepad. While under autonomous control, we need an equivalent way of measuring what is going on. A program that blindly drives motors at 70 per cent at all times may be able to negotiate a flat surface but may struggle to drive up a ramp. This kind of program contains an open-loop, but in order to compensate for this we need feedback. When we have a direct feedback-mechanism this is called a closed loop.

03 Pick your sensors

The most common way of measuring how far and how quick a robot is moving is through wheel or motor shaft encoders. These can be as simple as a paper disc attached to a motor shaft connected to a light sensor or a Hall Effect magnetic sensor.

By counting RPM and knowing the turns required to move a set distance, we

can start to compensate for different conditions. So when driving forward, instead of requesting 70 per cent motors we request 100rpm, and if the speed drops we increase the motor power.

04 Build an obstacle-avoiding behaviour

With any self-driving technology, the one thing we don't want is to cause a collision, whether that is with a priceless vase, the pet cat or a pair of slippers. The cheapest sensor we could find for this purpose is an Ultrasonic sensor (HC-SR04, we found one for about £1.99 from eBay). The HC-SR04 has a trigger and echo pin; set the trigger to high to emit a pulse, then immediately start counting until the echo pin goes from low to high. The time taken plus a small amount of maths, results in a distance between 2cm and 150-200cm.

05 Be quick or you'll miss it

The pulse emitted by the HC-SR04 sensor travels at the speed of sound, so we need to make sure the Pi's operating system is free and ready to measure the resulting echo. Unfortunately, Raspbian and the general flavours of GNU/Linux cannot guarantee us a real-time response. For this reason, we may want to measure more frequently or average the sensor readings over time. Either way, we cannot use the ExplorerHAT code library because it introduces too much computing latency. Instead we'll use the RPi.GPIO library directly, see the example class at **ultrasonic.py**, here:

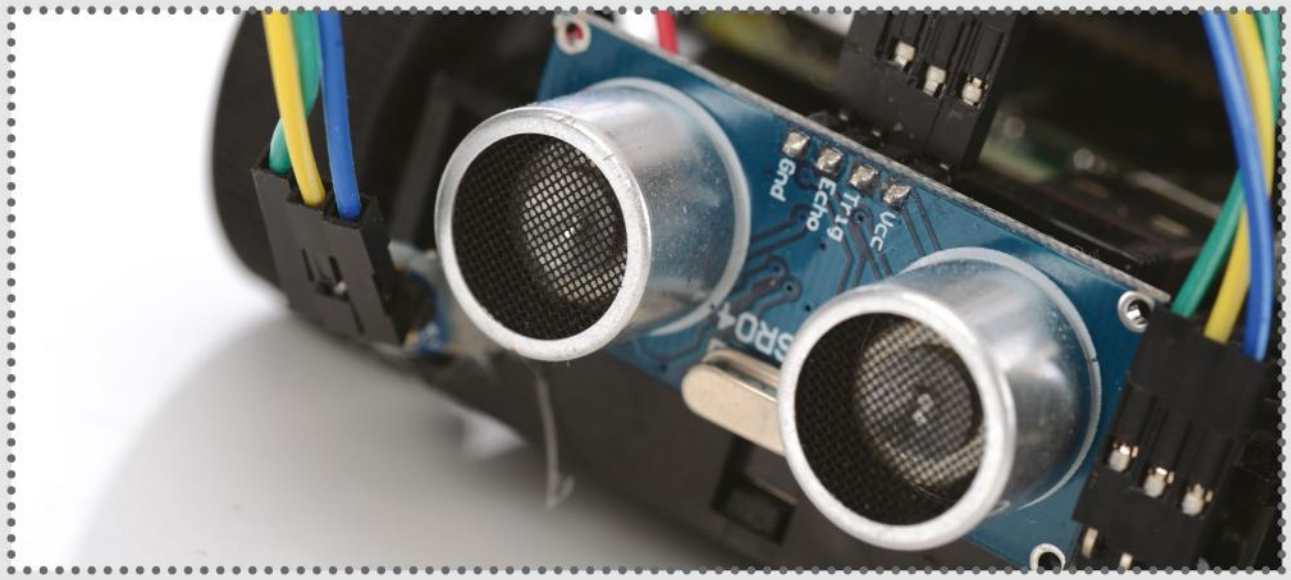
```
```
import RPi.GPIO as GPIO
import time

trigger = output 1 on ExporerHAT
echo = input 1 on ExporerHAT
```
```

```
 # For other output or
inputs consult http://
pinout.xyz
 class ultrasonic_
sensor:
  def __init__
(self,trigger,echo):
    self.GPIO_ECHO=echo
    self.GPIO_
TRIGGER=trigger

  def setup(self):
    GPIO.setmode(GPIO.
BCM)
    print “trigger = “ +
str(self.GPIO_TRIGGER)
    print “echo = “ +
str(self.GPIO_ECHO)
    # Set pins as output
and input
    GPIO.setup(self.GPIO_
TRIGGER,GPIO.OUT)  #
Trigger
    GPIO.setup(self.GPIO_
ECHO,GPIO.IN)      #
Echo

  def measure(self,
settleTime):
    # Set trigger to
False (Low)
    GPIO.output(self.GPIO_
TRIGGER, False)
```

```
    # Allow module to
settle
    time.sleep(settleTime)

    # Send 10us pulse to
trigger
    GPIO.output(self.GPIO_
TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(self.GPIO_
TRIGGER, False)
    start = time.time()
    while GPIO.input(self.
GPIO_ECHO)==0:
      start = time.time()

    while GPIO.input(self.
GPIO_ECHO)==1:
      stop = time.time()

    # Calculate pulse
length
    elapsed = stop-start

    # Distance pulse
travelled in that time
is time
    # multiplied by the
speed of sound (cm/s)
    distance = elapsed *
34000
```

```
    # That was the
distance there and back
so halve the value
    distance = distance
/ 2

    return distance
  def cleanup(self):
    GPIO.cleanup()
 ```
```

## 06 Don't crash into that wall

Now that we have a Python class which can take measurements through the ExplorerHAT, let's write a simple loop to read the values, then set up our robot to drive forward and halt if it detects anything around 30cm away or closer. For a code sample that only reads the values and prints them on the terminal use: **part4/distance/reader.py.** For the example below use: **part4/distance/dont_bump.py.** Also note that we are making the code self-documenting by naming variables such as: **safe_distance_cm, move_power_percent** and **sensor_**

## Shared autonomy

Sometimes a robot can be given full autonomy so it performs a task and regulates itself – think of a production line in a factory. At other times where the potential for an incorrect decision to be made is high, then we may want to share our autonomy with a robot so that we reduce the level of risk; a perfect example of this would be a self-driving car.
```



settle_time. Any number passed into a method without a clear name should be avoided. It is called a magic variable, because who knows what it means?

```
'''
import explorerhat
from ultrasonic import
ultrasonic_sensor

# trigger = output 1
# echo = input 1

# For other output or
inputs consult http://
pinout.xyz
trigger_pin = 23
echo_pin = 6

ultrasonic1 =
ultrasonic_sensor(echo_
pin, trigger_pin)
```

```
 ultrasonic1.setup()
 print("Ping..")
 while(True):
  sensor_settle_time =
0.1
  cm = ultrasonic1.
measure(sensor_settle_
time)
  safe_distance_cm = 30
  if(cm <= safe_
distance_cm):
   explorerhat.motor.one.
stop()
   explorerhat.motor.two.
stop()
  else:
   move_power_percent =
50
   explorerhat.motor.one.
forwards(move_power_
percent)
   explorerhat.motor.two.
forwards(move_power_
percent)

  print("%.1fcm" % cm)
```

07 Design the escape behaviour

Plug trigger into output 1 and echo into input 1 on the

ExplorerHAT. Connect positive to 5v and negative to GND. Once the robot has detected a distance of at least 30cm to an obstacle, then we need to act or it will remain stuck. A simple but effective escape behaviour is to back up a little and turn left, then let the main loop continue. Note that this is going to be an open loop so depending on the surface, the turn may be small or large. For now we will concentrate on smaller building blocks. See the full code in **part4/distance/roaming.py**.

```
  ```
  def escape():
    backup_time = 0.3
    turn_time = 0.15
    explorerhat.motor.one.
backwards(50)
    explorerhat.motor.two.
backwards(50)
    time.sleep(backup_
time)
    explorerhat.motor.two.
forwards(50)
    time.sleep(turn_time)
  ```
```

08 Create an emergency stop mechanism

Before we move on to look at a light-following behaviour, let's make sure we can stop the robot if it gets into trouble. You could do this by integrating the Wiimote from part three of the series and a Bluetooth dongle. The A button could pause the robot, and then perhaps B could resume the program after you have saved it from falling down a flight of stairs. The advantage of a hybrid mode is extra control and being able to operate outdoors without a laptop: this is called shared autonomy. The disadvantage of Bluetooth and a single USB port is that you have to swap over to the Wi-Fi dongle to update the code. For the rest of the tutorial, we will always keep a terminal open and hit **Ctrl+C** if things start to go wrong.

09 Mount your sensors

At this point you may be wondering how we can mount sensors onto the Zumo's tiny chassis. If you have access to a laser cutter or makerspace you may already have everything you need to make an extra layer for the robot. For our test rig, we have used hot glue to secure the ultrasonic sensor and two additional Adafruit ALS-PT19 Analog Light sensors from Pimoroni (£3.00 each). Make sure you only apply a small amount of glue and that you hold the sensor in place until dry so that it keeps the position you want.

10 Connect the ALS-PT19s

The ALS-PT19s are simple light sensors with built-in resistors. We just need to provide a positive and negative wire along with a third wire to read the signal. Plug one into analog 1 and the other into analog 2. There are only two

5v connectors and two GND connectors, so use a tiny breadboard to break these out and to allow us to power the ultrasonic sensor at the same time.

11 Read the analog values from the ALS-PT19s

Analog values can be read through the Pimoroni library and will give a value between 0.0-5.0 representing the input voltage. Here is a quick example of how to read the sensors.

```
print(explorerhat.analog.one.read())
print(explorerhat.analog.two.read())
```

The simplest control program would read both sensors and then immediately give more power to the left or right motors to initiate a steering motion. The best control programs would average out

the value to smooth it and perhaps use a PID controller mechanism to make sure the system doesn't over-correct. Here's our attempt; we've left lots of room for improvement.

```
 ```
 import explorerhat
 import time
 from ultrasonic import
*

 def get_
percentage(left, right):
  if(left > right):
   return (20, 50, "<")
  return (50, 20, ">")

 while(True):
  left = explorerhat.
analog.one.read()
  right = explorerhat.
analog.two.read()

  tolerance = 0.10
  delta = abs(left -
right)
  if(delta >= tolerance):
   percentage_power =
get_percentage(left,
right)
```
```

```
    explorerhat.motor.two.
forwards(percentage_
power[0])
    explorerhat.motor.one.
forwards(percentage_
power[1])
    print(percentage_
power[2])
  else:
    percentage_power =
(50, 50)
    print("^")
    explorerhat.motor.two.
forwards(percentage_
power[0])
    explorerhat.motor.one.
forwards(percentage_
power[1])

  time.sleep(0.1)
```

12 Construct the final robot control program

This is the final part of our tutorial and our final robot control program. It will roam around an environment making sure not to bump into anything and will seek out

Maker spaces near you

Maker and Hacker Spaces can be a great way to get in touch with like-minded people, and can also give you a way to access tools and machinery that would otherwise be outside of your budget. Raspberry Pi Jams are also good places to showcase your own handiwork. A great place to demo your robot would be at Cam Jam's PiWars event. Find it at: **https://piwars.org.**

the strongest light source. We were able to drive the robot around obstacles using a beam of light from a CREE LED torch with a beam that can be focused. By shining the torch on either side of the robot, it is possible to control its direction and course. The full code is provided in **part4/follow_light/follow_dont_bump.py.**

```
'''
import explorerhat
import time
from ultrasonic import *
ultrasonic1 = ultrasonic_sensor(6, 23)
ultrasonic1.setup()

bump_buffer_cm = 20

def get_percentage(left, right):
  if(left > right):
    return (20, 50, "<")
  return (50, 20, ">")

def escape():
  backup_time = 0.3
  turn_time = 0.15
```

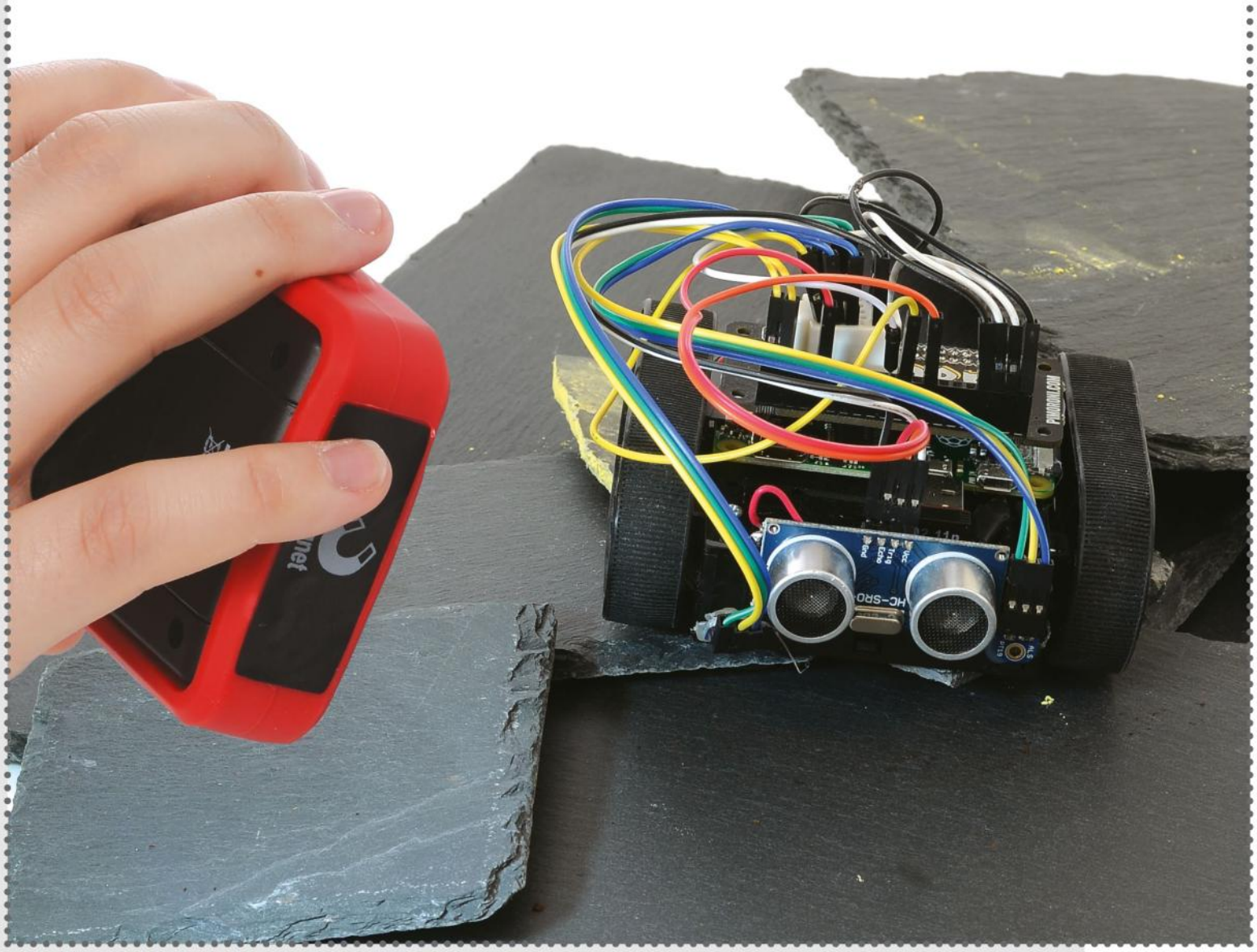
```
    explorerhat.motor.one.
backwards(50)
    explorerhat.motor.two.
backwards(50)
    time.sleep(backup_
time)
    explorerhat.motor.two.
forwards(50)
    time.sleep(turn_time)

  while(True):
    if(ultrasonic1.
measure(0.10) <= bump_
buffer_cm):
      explorerhat.motor.one.
stop()
      explorerhat.motor.two.
stop()
      escape()

    left = explorerhat.
analog.one.read()
    right = explorerhat.
analog.two.read()

    tolerance = 0.10
    delta = abs(left -
right)
    if(delta >= tolerance):
      percentage_power =
get_percentage(left,
right)
```

```
    explorerhat.motor.two.
forwards(percentage_
power[0])
    explorerhat.motor.one.
forwards(percentage_
power[1])
    print(percentage_
power[2])
  else:
    percentage_power =
(50, 50)
    print("^")
    explorerhat.motor.two.
forwards(percentage_
power[0])
    explorerhat.motor.one.
forwards(percentage_
power[1])

  time.sleep(0.1)
```

13 Decide what enhancements you want to make

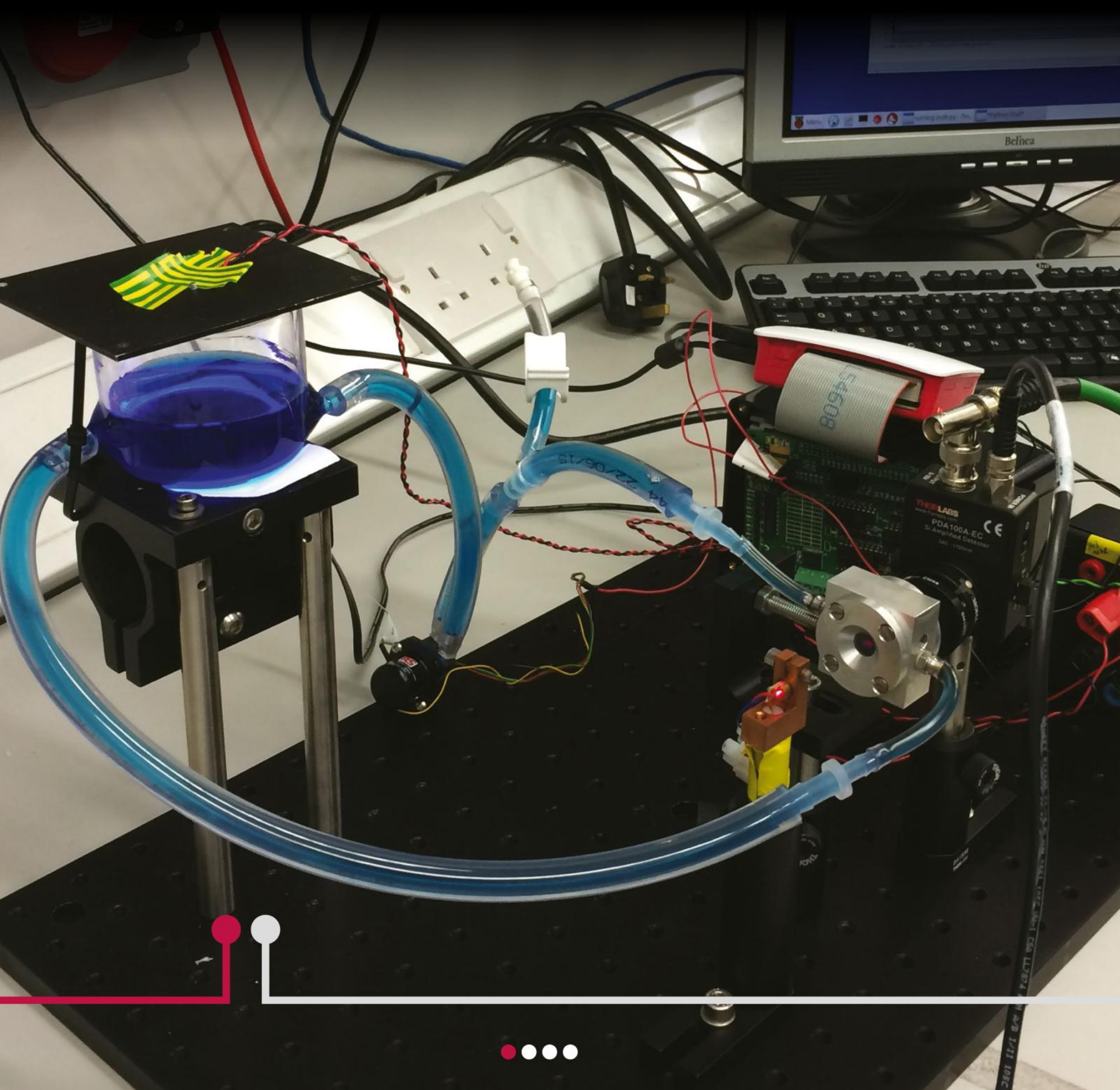
In the previous tutorial we looked at the advantages of using classes for development in Python. Perhaps you could separate each behaviour into individual classes? Maybe you would rather create a line-

following robot, in which case, switch out the sensors and with a few modifications the same code should be able to negotiate a simple course. We also mentioned magic variables. Can you spot any in the code examples? Maybe you could refactor them into separate variables with self-documenting names. Have fun with your robot and make sure to send us a picture of the finished product!



Producing clean water

Saving lives with the Raspberry Pi has never been easier thanks to Jessica Mabin's photocatalyst project



Could you tell us a little about what photocatalysts are and your drive for the project?

One in ten people don't have access to clean drinking water; including some of the most remote parts of the world. The water crisis has been named as the number one global risk based on impact to society, as quoted by the World Economic Forum in January 2015 [the WTF also rated it the third greatest threat for 2017]. This is a pressing issue and therefore has a wealth of potential research implications for both large-scale treatment plants and small-scale handheld filtration units to consider. Although these systems have been proven to clean waste water of harmful pollutants, they are costly and often only partially effective in the long run.

There is therefore demand for a system that not just filters the water, but removes all pollutants. A photocatalyst is a chemical that is activated with energetic photons to promote a reaction. In the case of this project, TiO2 (titanium dioxide) is activated by ultraviolet photons, which then facilitate two important chemical reactions: oxidation and reduction. These reactions produce highly reactive radicals that rip apart the chemical bonds in the pollutants, leaving just water and non-hazardous gases. One of the most important properties of catalysts is that they're not used up in reaction, making them more sustainable sources for continuous water treatment.

Could you tell us where the original idea came from?

My supervisor, Dr Stephen A Lynch, laid the foundations for this research when he was working at University College London. After moving to Cardiff University five years ago, a renewed interest led to my involvement, first as an MSc student and then as the topic of my PhD. The project has evolved from needing expensive dedicated test equipment costing thousands of pounds, to an equivalent test system

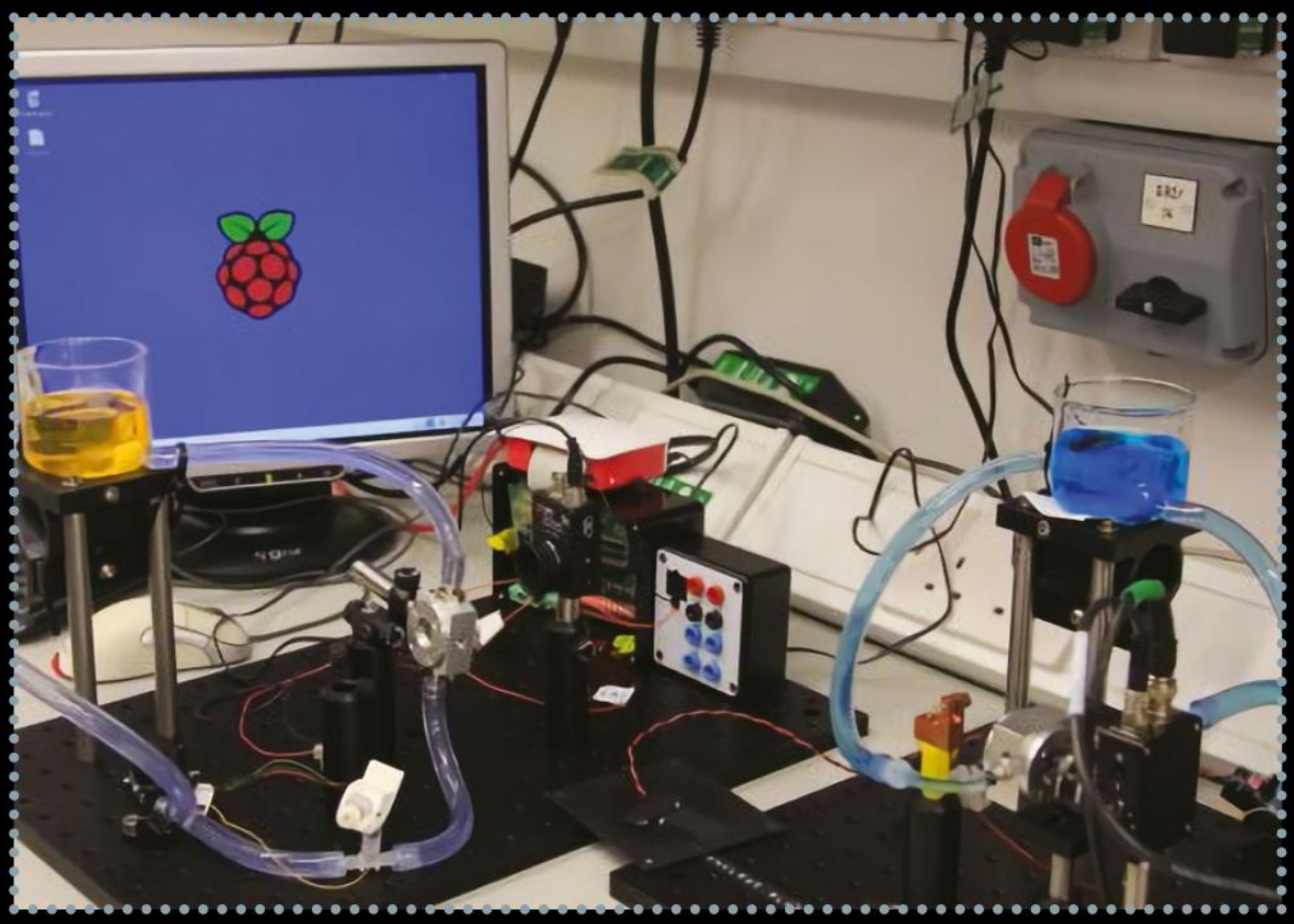
Jessica Mabin is a PhD student in Physics and Astronomy at Cardiff University. She has been looking at different ways to use science to help solve global, wider issues.

based around the Raspberry Pi, costing about £100 in total. This has enabled us to multiplex and run multiple experiments without having to spend too much money.

How was the Pi used and incorporated into this project?

We have developed the setup from using expensive multimeters and power sources to exploit the abilities of the Raspberry Pi, together with an expansion board containing ADC and DAC chips.

We decided to make this move as we wanted a portable detection system that could be inexpensively replicated on a wider scale. The experiment incorporates a laser diode that emits red light that passes through a flow cell and is measured on a photodiode. The voltage generated by the photodiode is digitised by the ADC chip and read by the Raspberry Pi. The ability to inexpensively replicate the setup allows us to perform experiments testing multiple variables in parallel. This multiplex advantage is very important as each experiment can take several hours, and of course, errors can happen.



THE PROJECT ESSENTIALS

Raspberry Pi
D.I.Y HAT
Photodiode
Multiple beakers
Expansion board with ADC and DAC chips
Wiring and tubing

Due to the advanced nature of the project, did you find any limitations with the Raspberry Pi?

We are only just scraping the surface regarding the full potential of the Raspberry Pi. With the appropriate expansion boards, we could exploit the Raspberry Pi to measure many more important environmental parameters. The number of bits on the ADC limits the sensitivity and resolution of the voltage measurement. Moving from a 16-bit ADC to a 24-bit ADC would certainly improve this. We are still improving the technology for coating surfaces with the TiO2 photocatalyst and this is materials research that's still ongoing but it is progressing well.

What are your plans moving forward? Do you feel there's more to uncover from using the Raspberry Pi in science and research?

We have not nearly tapped all of the resources that the Raspberry Pi can potentially give us. We envisage the it being the microcontroller behind a complete system. As well as measuring the pollutant concentration, the Raspberry Pi could, for example, be programmed to control the pumping speed and adjust this to match the rate at which the pollutants are being broken down. At the moment we are exploiting the capabilities of the Raspberry Pi to better understand the underlying material science of photocatalysis. We need to engineer and optimise the properties of the TiO2 photocatalytic material before it is robust enough to use in environments outside the lab. Ideally we would like to develop a standalone unit that we could test on real pollutants out in the

Get alerts with the Raspberry Pi baby monitor

Keep an eye on your little one while they sleep

While you're settling down to enjoy your latest boxset, there's always that nagging feeling at the back of your mind – is the baby asleep, or crying down the house? Surround sound tends to get in the way, which means we need to rely on baby monitors.

While audio monitors are useful, the trend these days is more towards video baby monitors, in particular those that display footage on an app or by opening an IP address in a mobile browser. Throw in some motion detection and alert software, and you've got an incredibly useful tool – you won't be surprised to learn, the Raspberry Pi can do for a fraction of the cost.

All you'll need is a Raspberry Pi (the newer the better), a USB webcam or the PiCam (the NoIR infrared version is even more suited to nighttime use), and a device to view the streamed footage on. You now don't need to be worried about baby again – just check your phone to see what they're up to.

THE PROJECT ESSENTIALS

Wireshark (www.wireshark.org)

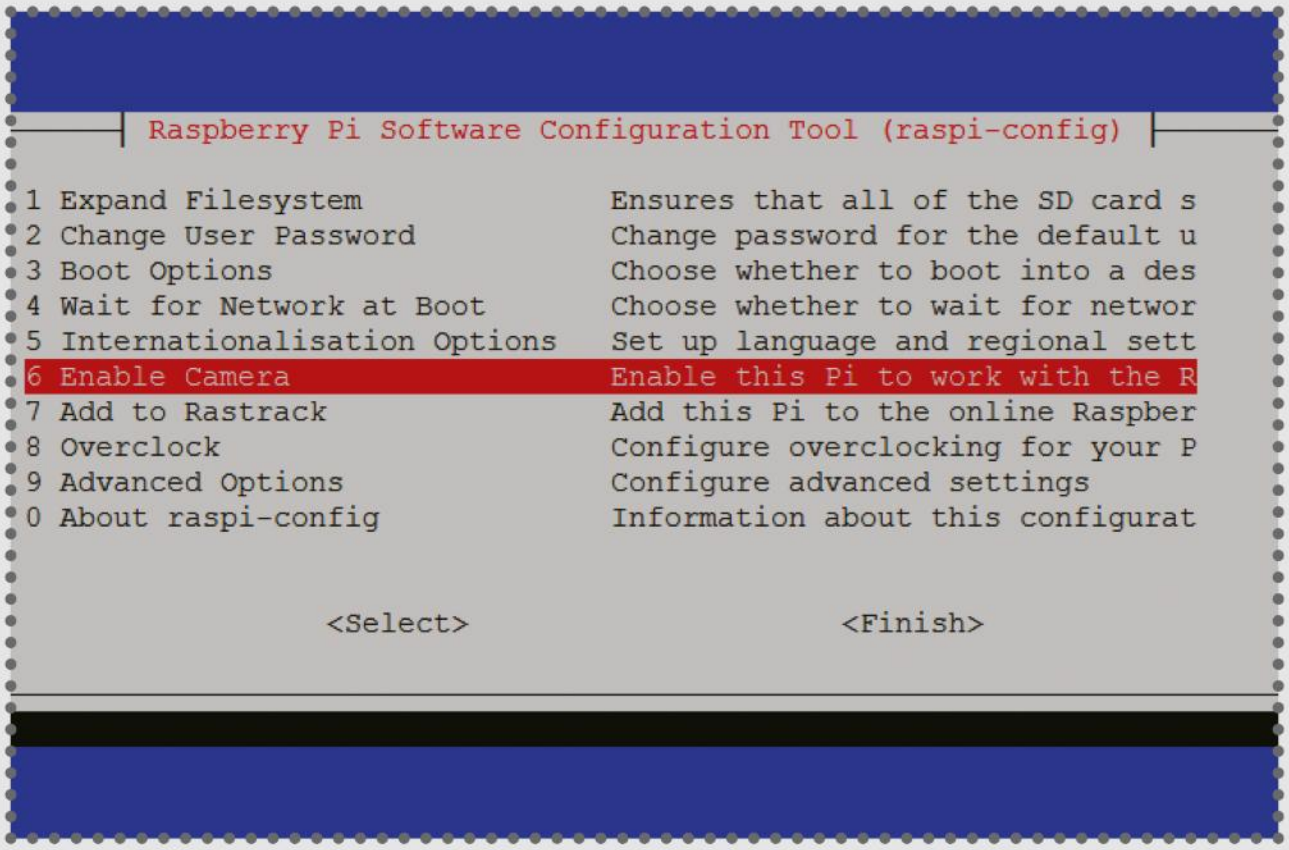
PiCam module or USB webcam

01 Survey the bedroom

Before you start installing and configuring your Pi, head to the baby's bedroom and take a look around. Where will you be placing the Raspberry Pi? Is it within reach of a power source? Do you need to connect an Ethernet cable, or is the wireless signal strong enough? It's vital at this stage to spend the necessary time planning the Pi's position in relation to different power sources and network connectivity.

02 Enable the camera

If you're using the PiCam, this will be disabled by default. You can enable this in the raspi-config tool. This can be accessed in the GUI by opening Menu > Preferences >



Left Remember to enable your camera so you can keep an eye on the little one

Raspberry Pi Configuration, where you should select the Interfaces tab and switch Camera to Enabled. Alternatively, run sudo raspi-config and choose option 6, Enable Camera. For USB webcams, use

```
sudo apt-get install fswebcam
```

03 Test the camera

You don't want to set this project up to find that the camera doesn't actually work. Whether you're using a USB webcam or the PiCam, you'll need to run a command to test the camera.

To confirm, use the GUI to browse to /home/pi and view the image.jpg file. This is preferable to checking in the command line, as this way you can ensure the image is not corrupt.

04 Install motion

The motion capture software, motion, can be installed after an update and upgrade of the Raspbian OS.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install motion
```

Older versions of motion will not start automatically, and display the "Not starting motion daemon" error message. To avoid this, you need to make sure you run the update and upgrade commands. If you're using the PiCam module, you'll also need a driver.

```
pi@raspberrypi:~ $ sudo apt-get install motion
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libmysqlclient18 mysql-common
Suggested packages:
  mysql-client postgresql-client
Recommended packages:
  ffmpeg
The following NEW packages will be installed:
  libmysqlclient18 motion mysql-common
0 upgraded, 3 newly installed, 0 to remove and 2 not upgraded.
Need to get 920 kB of archives.
After this operation, 4,101 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

05 Activate PiCam driver

To activate the PiCam driver, you need to enter the following command:

```
sudo modprobe bcm2835-v4l2
```

This enables the PiCam to communicate with third party apps, such as motion. However, you'll need to invoke the driver every time you reboot, unless you add it to rc.local. Open the file in nano:

```
sudo nano /etc/rc.local
```

Find an empty line before exit ="0" and enter:

Which camera?

Raspberry Pi projects that rely on a camera can usually work with a standard USB camera if you don't own a PiCam. However, the Pi's own camera will work without much trouble – conversely, you may have difficulty with a USB webcam if the drivers aren't available. It really depends on the project. The PiCam isn't the easiest device to position, whereas you'll enjoy more positioning flexibility with a USB webcam with a long cable.

```
modprobe bcm2835-v4l2
```

Then CTRL+X to exit, and Y to save.

06 Auto-start motion

Begin by opening the motion configuration file:

```
sudo nano /etc/default/motion
```

Here, we need to instruct the software to start each time the Raspberry Pi boots. Find the value "daemon off" and change it to read:

```
daemon on
```

```
# Rename this distribution example file to motion.conf
#
# This config file was generated by motion 3.2.12+git20140228

############################################################
#Daemon
############################################################

# Start in daemon (background) mode and release terminal (default: off)
daemon on

# File to store the process ID, also called pid file. (default: not defined)
process_id_file /var/run/motion/motion.pid

############################################################
# Basic Setup Mode
############################################################

                              [ Read 731 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Hit CTRL+X to exit, tapping Y to confirm you wish to save the file, and Enter to continue.

Next, confirm the motion daemon works on a reboot by restarting your Pi

```
sudo reboot
```

07 Configure motion

The next step is to configure the motion software. This means setting the frame-rate (how often an image is captured), image dimensions (larger images will take up more resources, thereby slowing the monitor) and setting the video format.

How you configure motion really depends on which model of Raspberry Pi you will be using for this project. First generation devices will still handle low-resolution images comfortably; for a hi-res feed, you should probably use the Raspberry Pi 3.

08 Edit the config file

```
 GNU nano 2.2.6          File: /etc/motion/motion.conf

norm 0

# The frequency to set the tuner to (kHz) (only for TV tuner cards) (default: 0)
frequency 0

# Rotate image this number of degrees. The rotation affects all saved images as
# well as movies. Valid values: 0 (default = no rotation), 90, 180 and 270.
rotate 0

# Image width (pixels). Valid range: Camera dependent, default: 352
width 640

# Image height (pixels). Valid range: Camera dependent, default: 288
height 480

# Maximum number of frames to be captured per second.
# Valid range: 2-100. Default: 100 (almost no limit).
framerate 2


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

To begin configuration, open **motion.conf**

```
sudo nano /etc/motion/motion.conf
```

Use the CTRL+W shortcut to open search, and find each of the following conditions, adding the values as specified:

```
daemon on
framerate 2
width 640
height 480
ffmpeg_video_codec mpeg4
stream_localhost off
control_localhost off
```

If you're recording in a darkened room, you might wish to adjust the brightness and contrast values.
With the changes made, hit CTRL+X to exit, confirming with Y and Enter.

09 Assign ownership to target directory

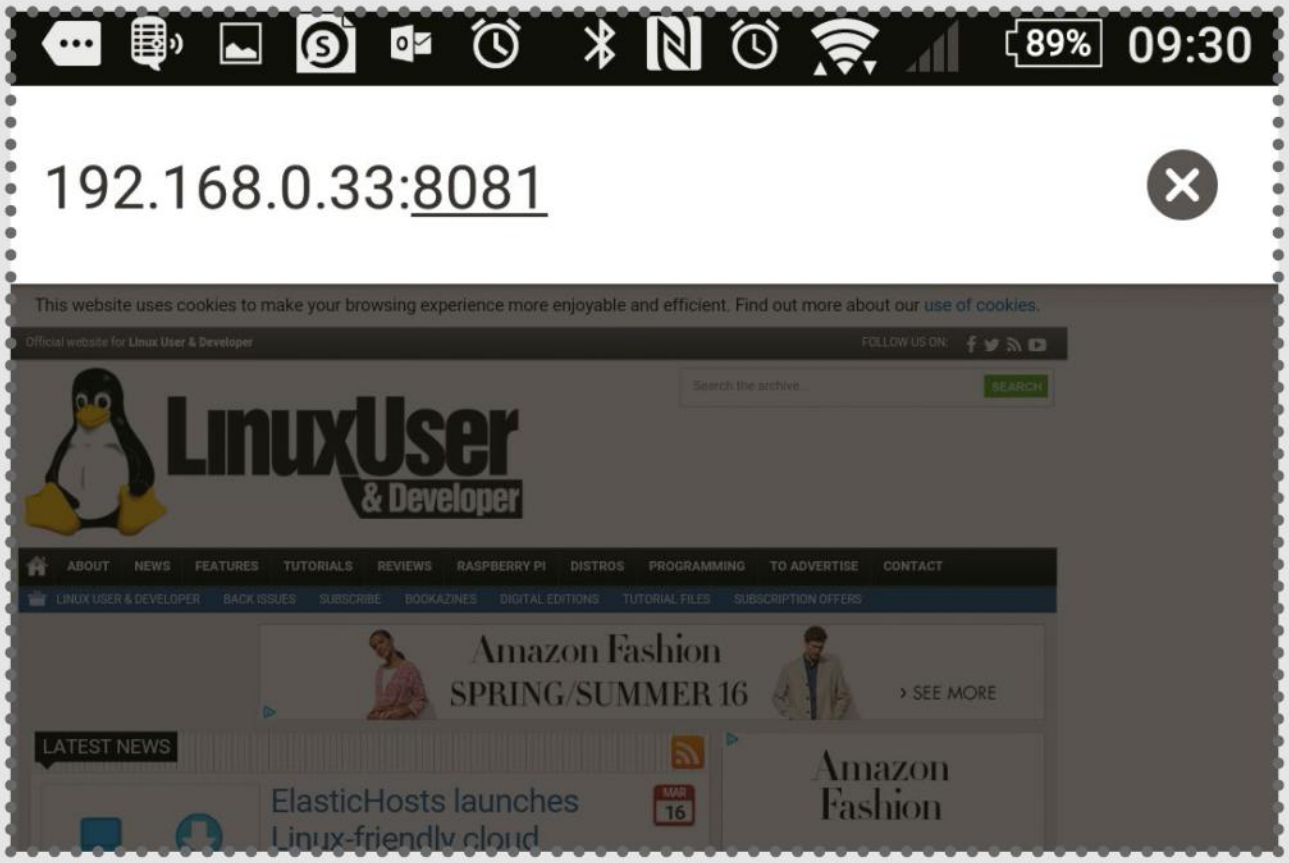
You may find that the camera stops streaming images after a short time. This is a permissions issue, one that causes a few images to appear on your screen before the whole thing times out.
Overcome this with:

```
sudo chown motion: /var/lib/motion
```

You can also set a custom file path in motion.conf. Look for target_dir and change as appropriate. Remember to save the file and restart motion when you're done.

10 Start and test your baby monitor

We now almost have a fully working baby monitor. At this stage, all that is left to get the baby monitor up and running is to launch motion:




```
sudo service motion start
```

You should be able to browse to the feed by entering the IP address of your Raspberry Pi in a mobile browser: `192.168.0.10:8081`

Check this on multiple devices on your network to confirm that it works.

```
  GNU nano 2.2.6         File: /etc/motion/motion.conf


# Number of frames to skip before each roundrobin step (default: 1)
roundrobin_skip 1

# Try to filter out noise generated by roundrobin (default: off)
switchfilter off


############################################################
# Motion Detection Settings:
############################################################

# Threshold for number of changed pixels in an image that
# triggers motion detection (default: 1500)
threshold 1500

# Automatically tune the threshold down if possible (default: off)
threshold_tune off

                        [ Search Wrapped ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

SSH and VNC

As with any project like this, you should forget about hooking up a keyboard and mouse and rely on SSH and/or VNC.

SSH is now enabled by default in Raspbian (check raspi-config otherwise) and makes connecting to the Pi's command line very simple. Meanwhile, VNC is one of several ways to display the Raspberry Pi desktop on your main PC. It can be set up by installing tightvncserver on your Pi, and running TightVNC Viewer on the main computer.

11 Adjust motion detection

It's unlikely that you will find that motion detection works right away. In order to adjust this for the environment you have the Pi baby monitor set up in, open

```
sudo nano /etc/motion/motion.conf
```

and use CTRL+W to search around for the 'Motion Detection Settings'.

Here you'll find various conditions with values that you can adjust, such as threshold and area_detect_ value. These will require some patient tweaking for the best results.

12 Make motion beep

To aid in tweaking the detection, you can enable a beep to sound when movement is captured. As a project like this needs some calibration to get the best results, this is a useful feature.

Again, this setting is found in motion.conf. Search for "quiet on" and change the setting to read

```
quiet off
```

Remember to undo this when you're happy with the movement capture, as it may disturb the little one.

13 Adjust image quality

While you can alter dimensions of the images captured by the PiCam board, it's important to be careful with the figures you enter in motion.conf. For instance, a dimension of 133x255 pixels probably won't work. Dimensions need to be multiples of 4. For larger options,

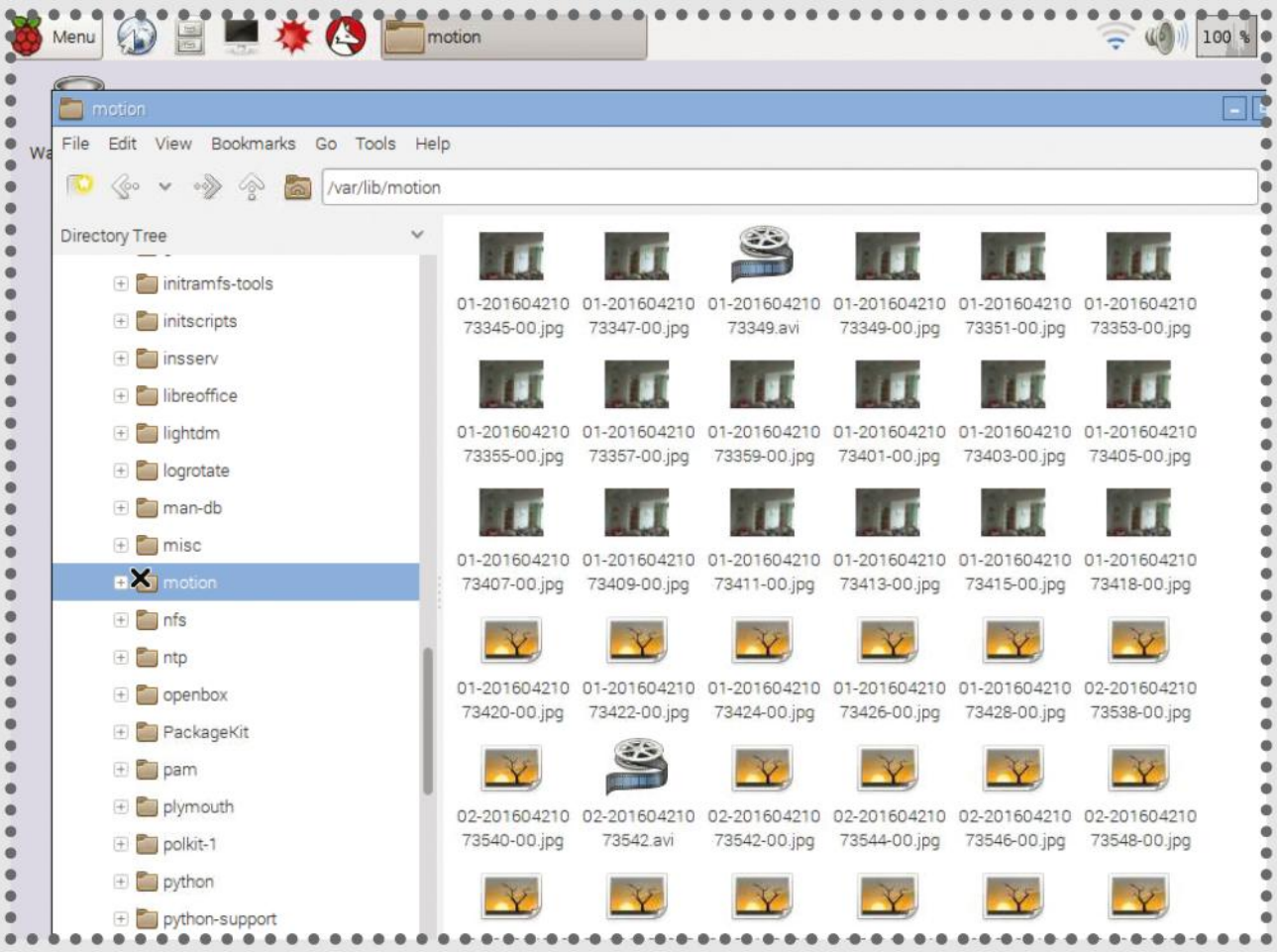
USB webcam and power

Different USB webcams have their own power requirements. Some will run off the Raspberry Pi's own power, others will require a power source, best provided via a powered USB hub. If you're positioning the Raspberry Pi baby monitor away from a power source, a rechargeable battery pack should be able to provide enough juice for both devices to last for around 12 hours.

look at 1280x800 or 1920x1080.
Not only will larger images impact bandwidth, they'll make the resulting AVI file larger.

14 Check saved images

To confirm the quality of the images captured by the Raspberry Pi baby monitor, boot into the GUI (or install tightvncserver and remote connect) and browse to /var/lib/motion to see how they are turning out. This should give you the info you need to adjust the dimensions of the captured images.



15 Troubleshoot camera connectivity

There's no guarantee that motion will work straight away – if no images are found, or you cannot connect to the stream (or both) it's worth running

```
tail -f /var/log/syslog
```

and

```
dmesg | tail
```

This will display any issues that the process is currently having, which is intended to (and hopefully will) help you diagnose and resolve any problems. Most issues will be driver-related, so keep this in mind with USB webcams. Press CTRL+Z to end.

```
pi@raspberrypi:~ $ dmesg | tail
[   18.819042] cfg80211:   (57000000 KHz - 66000000 KHz @ 2160000 KHz), (N/A, 40
00 mBm), (N/A)
[   22.113760] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
[   22.114488] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[  207.965707] media: Linux media interface: v0.10
[  207.992283] Linux video capture interface: v2.00
[  208.062384] bcm2835-v4l2: scene mode selected 0, was 0
[  208.068577] bcm2835-v4l2: V4L2 device registered as video0 - stills mode > 12
80x720
[  208.076533] bcm2835-v4l2: Broadcom 2835 MMAL video capture ver 0.0.2 loaded.
[  322.391108] bcm2835-v4l2: error 0 waiting for frame completion
[ 2682.082108] bcm2835-v4l2: error 0 waiting for frame completion
pi@raspberrypi:~ $
```

16 Name your images

Images collected by the motion software can be configured with a specific naming convention, based on date and time.

You can find these listed under "target base directory" in motion.conf. For instance, you can specify a folder for new images, based on date:

```
%Y_%m_%d/%v-%Y%m%d%H%M%S-%q1
```

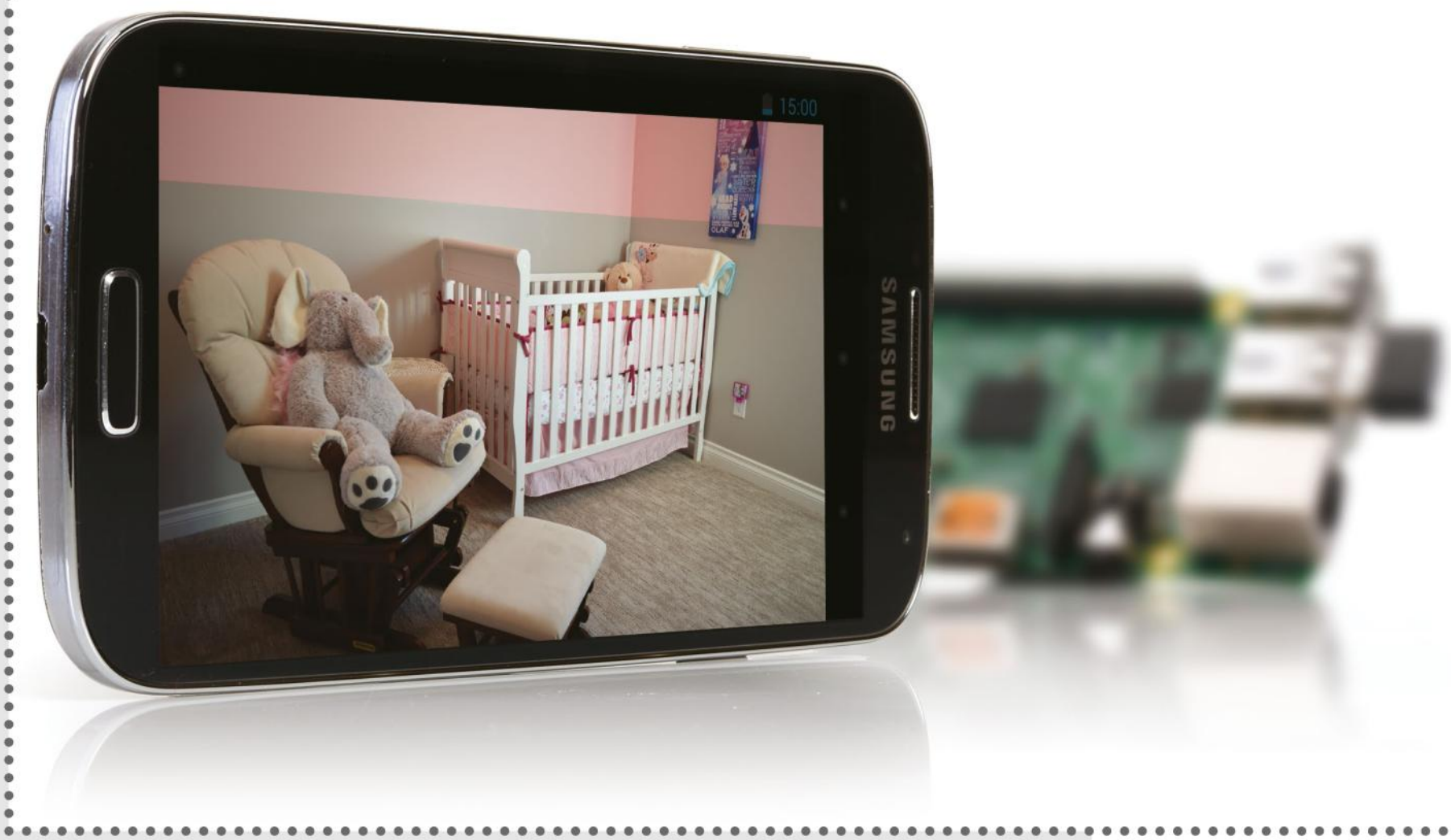
Note the "/" – both directory and images will be named according to date, with images also labelled with the timestamp.

17 Go beyond your home network

Wouldn't it be great to monitor your child's sleep from your favourite restaurant? You can do this by installing the No-IP software on your Raspberry Pi.

This software enables you to get around the fact that your ISP won't give you a dedicated IP address without paying a hefty premium, by installing a client app that enables you to view the baby monitor outside your home network.

Head to **www.noip.com** and check their knowledge base for details.



Turn a Pi into a router

Learn the basics of OpenWRT using a Raspberry Pi as a router

Controlling the interconnects between various devices is paramount to keeping systems secure and safe. Sadly, most router operating systems are closed source – finding vulnerabilities in them is difficult to impossible. Further, running dedicated open-source router operating systems is not a solution for the average user, as they tend to demand high-end hardware with prohibitively high prices.

OpenWRT is an affordable and efficient alternative. It foregoes some of the complexities found in traditional router operating systems, thereby allowing for lower hardware requirements. The community has ported the system to various routers: with a little care, a compatible router can be found for £100. Invest a few hours to transform it into a lean and mean fileserver, torrent client or – configuration allowing – even a system capable of controlling real-world hardware via serial links.

Here we will introduce you to the basics of OpenWRT using a well-known single-board computer. That knowledge can then be applied to a variety of other, more suitable hardware solutions.

THE PROJECT ESSENTIALS

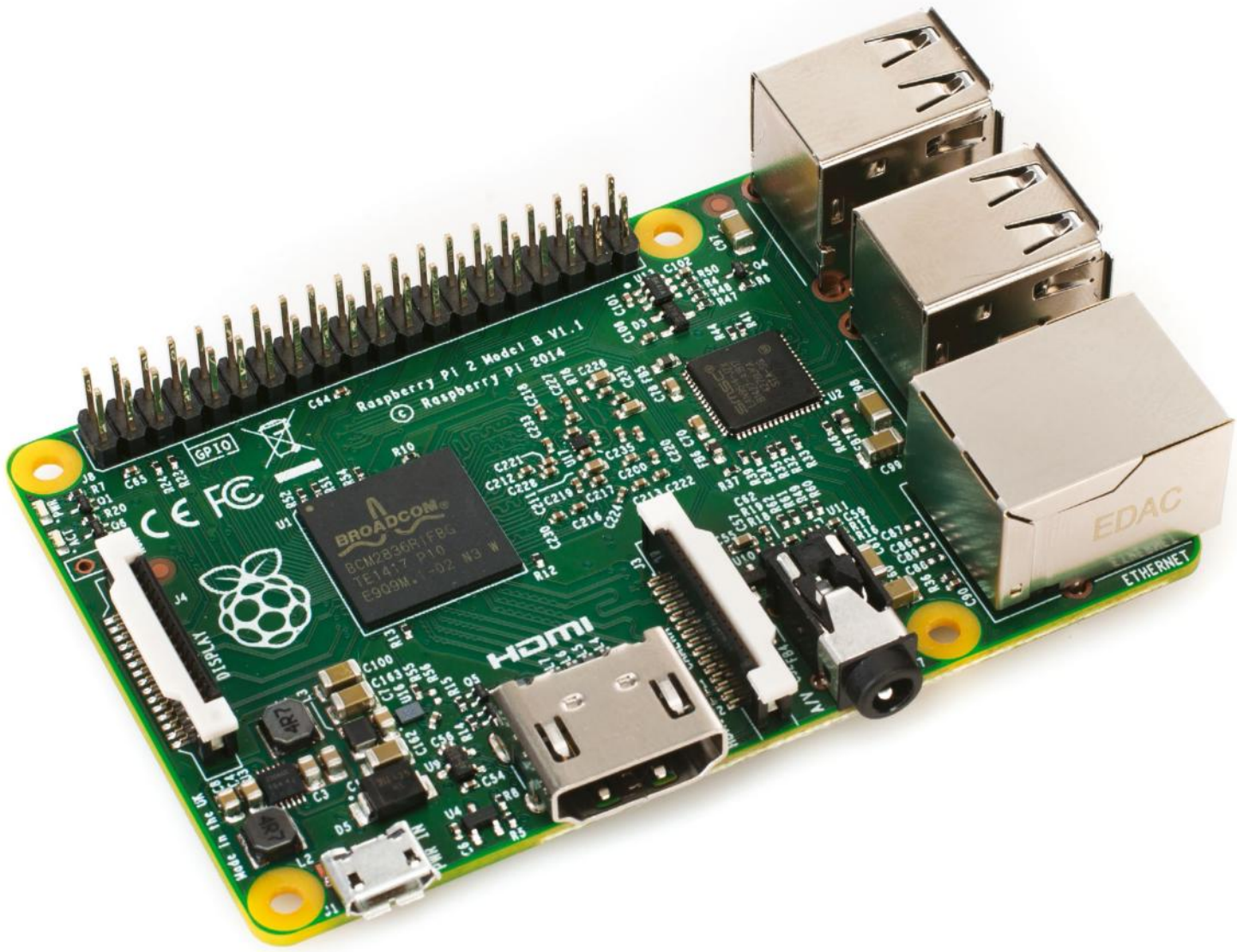
Raspberry Pi (V2B recommended, V1B possible)

Decent quality MicroUSB power supply

Cardreader + MicroSD Card

Compatible USB-Ethernet adapter (i-tec-europe.eu/?t=3&v=296)

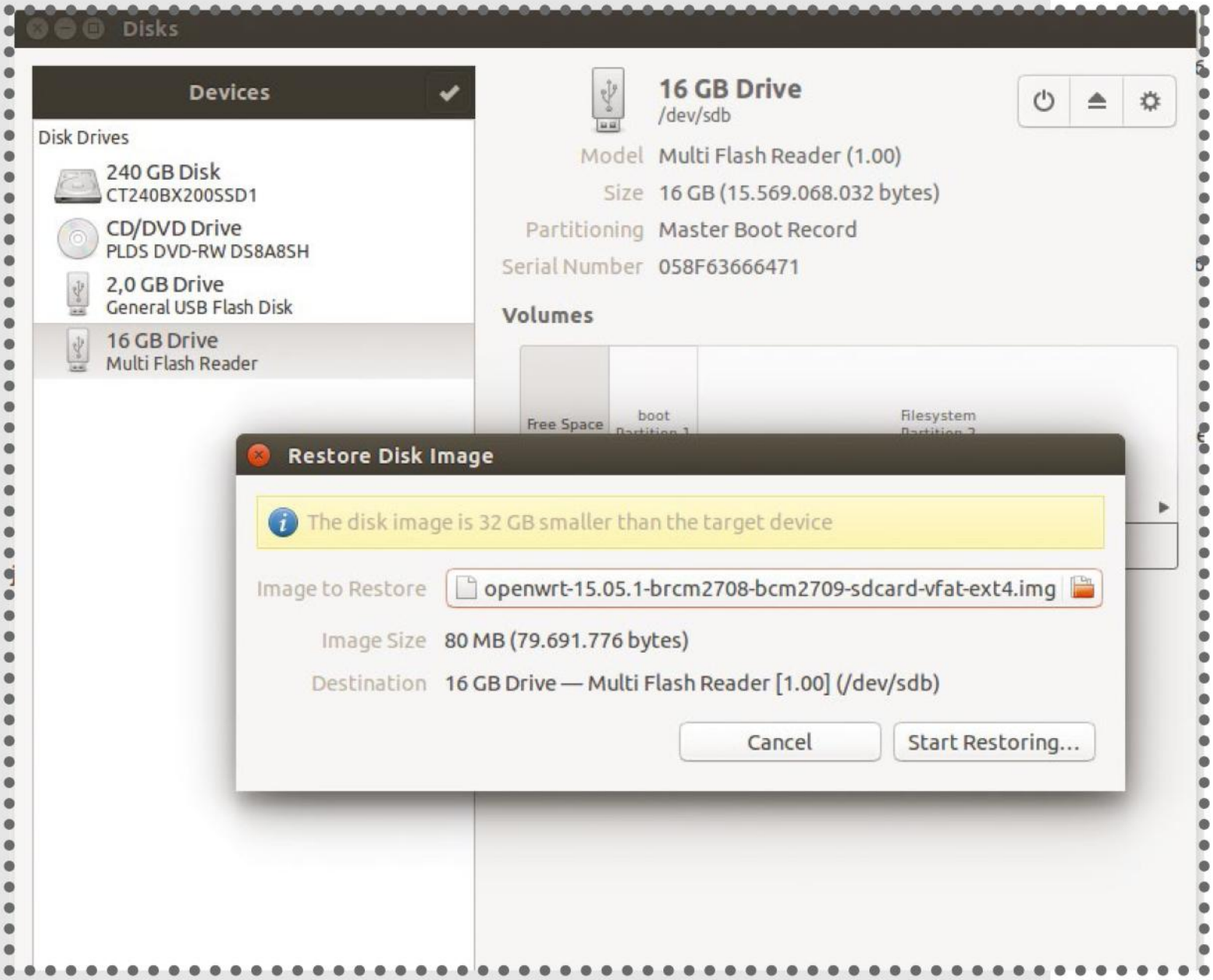
Optional, but recommended: Ethernet switch (no router!)



01 Set it up

Deploying an operating system requires you to be in possession of a suitable image: due to differences in the hardware, RPi 1 and 2 are targeted with different files which can be downloaded at **http://bit.ly/1T7t4UC**. The following steps are performed on a Raspberry Pi 2 using Chaos Calmer 15.05.1.

Burn the image 'openwrt-15.05.1-brcm2708-bcm2709-sdcard-vfat-ext4.img' to the SD card in a fashion of your choice: Ubuntu's Image Writer is the utility shown in the figure. Finally, insert the SD card, connect the RPi's native ethernet port to your PC and power up the contraption. Interested individuals can connect a HDMI monitor in order to see the boot process 'live'.



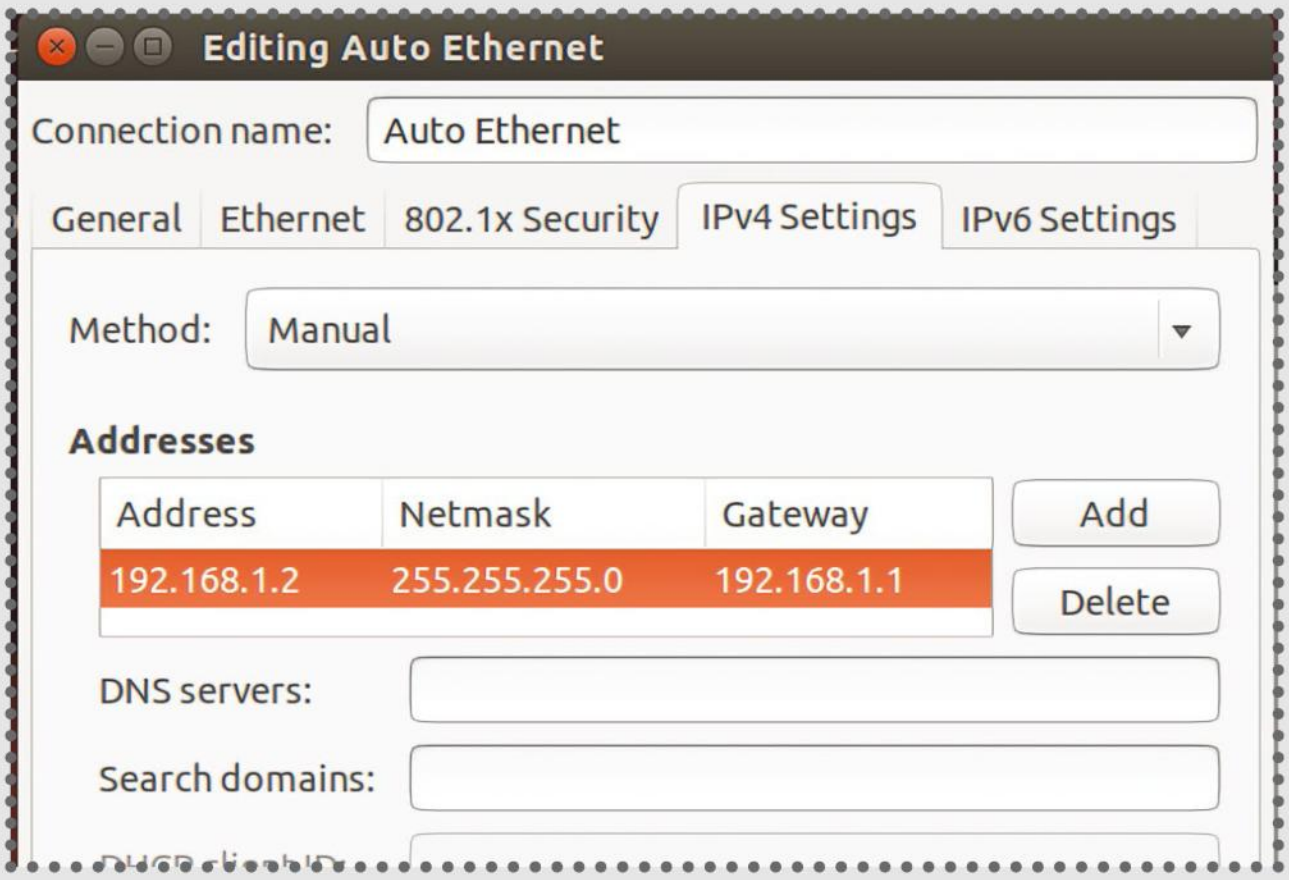
02 Get connected

Starting OpenWRT on a Raspberry Pi 2 takes about 30 seconds. When done, the message shown in the figure will appear. At this point, the ethernet port of the Raspberry Pi 2 will be set to a fixed IP address of 192.168.1.1 and will await network connections from other workstations. Open the 'Network connections'

```
Please press Enter to activate this console.
[    8.852972] NET: Registered protocol family 10
[    8.881403] ip6_tables: (C) 2000-2006 Netfilter Core Team
[    8.901576] hidraw: raw HID events driver (C) Jiri Kosina
[    8.930744] ip_tables: (C) 2000-2006 Netfilter Core Team
[    8.946261] nf_conntrack version 0.5.0 (14892 buckets, 59568 max)
[    9.028739] usbcore: registered new interface driver usbhid
[    9.038935] usbhid: USB HID core driver
[    9.055374] xt_time: kernel timezone is -0000
[    9.070444] PPP generic driver version 2.4.2
[    9.080318] NET: Registered protocol family 24
[   11.308055] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
[   11.323760] device eth0 entered promiscuous mode
[   11.336194] IPv6: ADDRCONF(NETDEV_UP): br-lan: link is not ready
[   21.231381] random: nonblocking pool is initialized
```

applet of the host, and configure it to use a static IP address via the settings shown in the figure.

Be aware that 192.168.1.1 is a popular address for routers: if your Wi-Fi router uses it too, then the network connection needs to be disabled during the following steps.



03 Telnet or SSH?

Chaos Calmer 15.05.1 keeps the Telnet service open on unconfigured instances. The first bit of work involves connecting to the Telnet client.

Invoke the passwd command to set a new password. Complaints about low password strength can be ignored at your own peril: passwd will not actually prevent you from setting the passcode to be whatever you want, but hackers might be delighted about the easier attack vector. Once the new root password is set, the Telnet server will disable itself.

Limited support for Raspberry Pi 3

On paper, the RPi 3's embedded Wi-Fi module makes it a perfect access point. However, this is not the case for two reasons. First of all, the range of the module has been shown to be abysmal in lab tests. Second, BroadComm has not released the driver code – at the time of going to press, its use in OpenWRT is unsupported.

From that moment onward, your OpenWRT instance can only be controlled via ssh.

```
tamhan@tamhan-thinkpad:~$ telnet
192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
. . .
root@OpenWrt:/# passwd
Changing password for root
New password:
  Bad password: too short
Retype password:
Password for root changed by root
- - -
tamhan@tamhan-thinkpad:~$ ssh
root@192.168.1.1
The authenticity of host '192.168.1.1
(192.168.1.1)' can't be established.
RSA key fingerprint is 11:80:4b:14:cc:b8:9a:
a6:42:6a:bf:8d:96:2a:1b:fa.
Are you sure you want to continue
connecting (yes/no)? yes
Warning: Permanently added '192.168.1.1'
(RSA) to the list of known hosts.
```

04 Let's play nice

The following steps assume that your router will live behind another router. As the activation of USB support requires the downloading of a batch of packages, our first act involves making OpenWRT play nicely with the rest of the network. As the stock

Pis make bad routers

Even though the Raspberry Pi makes a great demo and evaluation system, using it in practice might lead to suboptimal performance. This is caused by the unique bus architecture. Both ethernet ports must share the USB bandwidth. On the RPi 2, this problem is mitigated by the significantly higher CPU performance. For large networks, using an X86 based embedded system tends to yield better results. Single-board computers like the BananaPi are another alternative, but tend to crash when confronted with specific ethernet packages.

distribution includes only vi, open the web interface by entering http://<ip> into a computer of your choice. Next, click Network>Interfaces and click the Edit button next to br-lan. Set the protocol field to DHCP client and select the Switch Protocol button.
Finally, click Save&Apply, close the web page and disconnect the RPi from your PC. Next, connect both PC and Pi to the existing router and run nmap as root in order to find its newly-assigned IP address.
The command shown here is a little nifty in that it instructs nmap to scan the entire 255 addresses of the subnet – be sure to adjust it to your local environment. Furthermore, keep in mind that the IP settings of the PC must be restored to the ones used originally, with a reboot recommended for good practice.

```
tamhan@tamhan-thinkpad:~$ sudo nmap -sn
192.168.1.0/24
 Starting Nmap 6.40 ( http://nmap.org ) at
2016-05-03 21:14 CEST
 . . .
 Nmap scan report for 192.168.1.104
 Host is up (-0.099s latency).
 MAC Address: B8:27:EB:53:4E:D9 (Raspberry
Pi Foundation)
```

05 Deploy missing USB drivers

At this point, our OpenWRT instance is connected to the internet at large. This allows opkg to download required packages. Connect yourself using SSH and the IP address determined by NMAP, and proceed to downloading the packets listed in the

code accompanying this step. When all modules are installed, entering dmesg will show that the ASIX ethernet interface has been detected and configured as interface eth1 according to the figure.

```
opkg update
opkg install kmod-usb2 usbutils kmod-usb-core
opkg install kmod-usb-net kmod-usb-net-asix
```

06 Connect

Even though dongles based on the ASIX AX88772B are quite common, not being able to procure one does not condemn your experiment to failure. Connect the USB to LAN bridge to a Raspberry Pi running Raspbian and enter the lsmod command. It will provide you with information about the driver modules being used, which can then be tracked down on OpenWRT. Googling <chipset> openwrt or <productname> openwrt can also yield useful results.

07 Open the web interface

After completing the kernel configuration process, our new interface is ready and awaits the deployment of a configuration. As the OpenWRT image provided for the Raspberry Pi restricts us to vi (nano will not install), configuration is best done via the web interface we touched on earlier. It can be accessed by pointing your browser at the URL of the router; log-in can be accomplished via the root password used on the command line.

08 Let's get routing

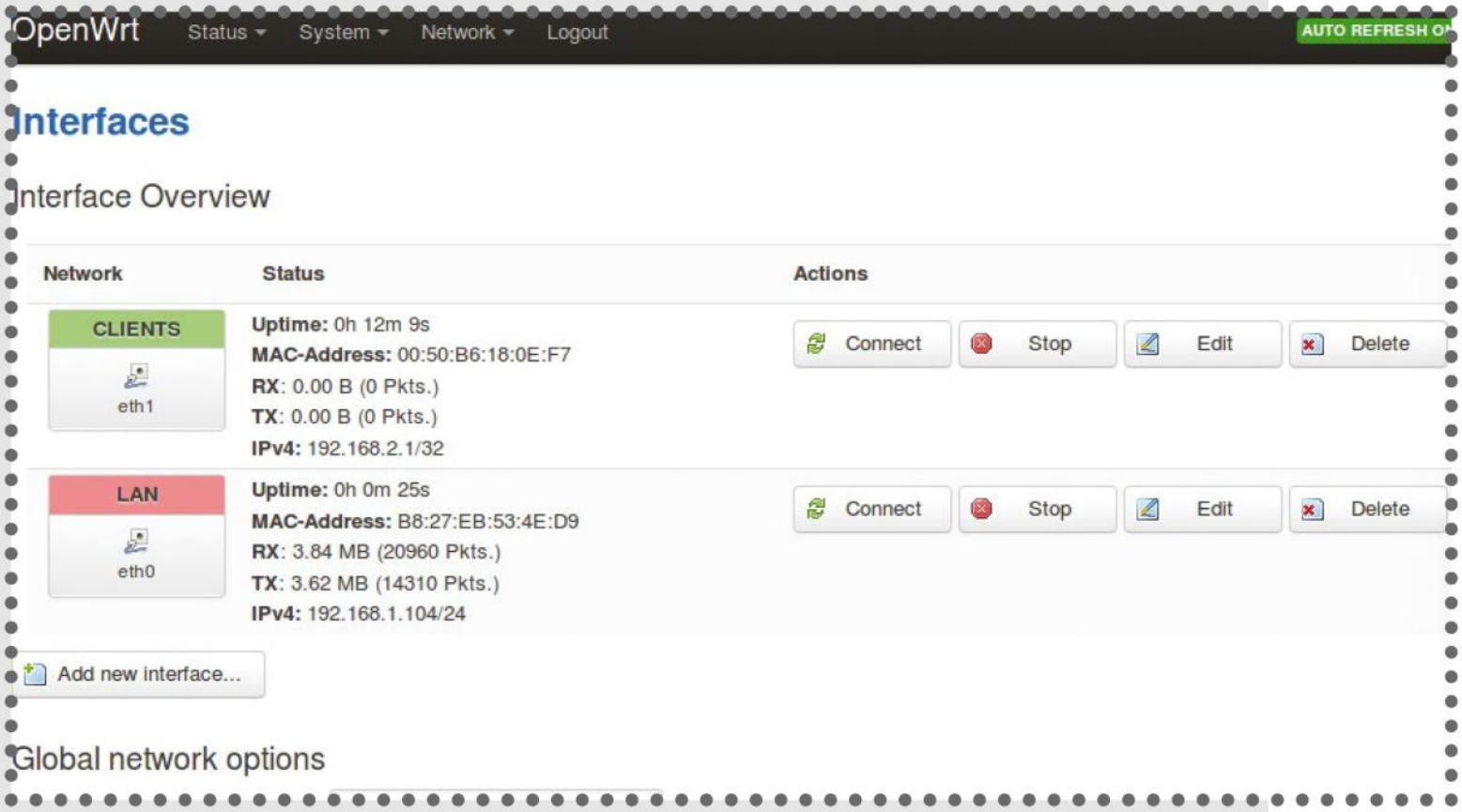
The newly-created USB ethernet port will be used to connect clients. You can connect either a 'dumb switch' or a single device. In both cases, a DHCP server is needed in order to provide IP addresses to the clients.

Click the Add new interface button, and name the new Interface Clients. Next, select the protocol to be the Static address and select the newly created interface eth1. Next, scroll to the bottom of the window and click the Setup DHCP Server button in order to fully populate the form.

With that, the IPv4 address and broadcast fields must be set up. Finally, click Save & Apply in order to commit the changes to the network stack. Next, open the network configuration once again and set the Firewall Settings to the firewall zone LAN.

09 Rearrange the interfaces

By default, the LAN interface is bridged. This is not necessary. Open its properties, select the Physical



Settings tab and unselect the Bridge interfaces checkpoint. Next, open the Firewall settings tab and assign the WAN zone. Finally, another click on Save & Apply makes OpenWRT assign the attributes leading to the configuration.

10 Firewall ahoy!

From this point onward, attempting to interact with the LuCI frontend from 'outside' of the network will lead to 'Unable to connect' errors – by default, remote configuration is not allowed to make attacks on OpenWRT more difficult.

Solve this problem by disconnecting the workstation from the "outer router", and connect to the Raspberry Pi's USB network interface instead. Then, perform an ifconfig command and connect to the standard gateway in order to open the LuCI interface again. Should you find yourself in the situation that no IP adress is assigned to the workstation, reboot the process computer and reconnect the ethernet cable.

```
tamhan@tamhan-thinkpad:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr
28:d2:44:24:4d:eb
  inet addr:192.168.2.157  Bcast:192.168.2.255
Mask:255.255.255.0
  inet6 addr: fe80::2ad2:44ff:fe24:4deb/64
Scope:Link
```

11 Test the presence of the router

As long as all other network connections are disabled, the workstation can connect to the internet only via the RPi. Enter "mtr www.google.com" in a

command line in order to generate the tree structure – from a latency point of view, our OpenWRT access point looks quite good when operating under a lighter load.

12 Analyse the network status

Generating live diagrams with further information about the state of the router is an interesting feature. Open LuCI and select Status > Realtime graph in order to open a set of diagrams telling you more about CPU and network loads.

```
tamhan@tamhan-thinkpad: ~
                          My traceroute  [v0.85]
tamhan-thinkpad (0.0.0.0)                          Wed May  4 00:03:50 2016
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                        Packets               Pings
 Host                                 Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 192.168.2.1                        0.0%   135    0.5   0.4   0.3   0.7   0.0
 2. 192.168.1.1                        0.0%   135    0.9   0.9   0.8   1.1   0.0
 3. 192.168.0.1                        0.0%   135    1.0   1.0   0.9   1.6   0.0
 4. ???
 5. 84.116.25.33                       0.0%   135   15.8  11.2   7.6  29.2   4.2
 6. at-vie15a-rd1-ae31-2048.aorta.net  0.0%   134   41.6  23.1  19.3  56.3   5.0
 7. 213-46-160-77.aorta.net            0.0%   134   33.3  38.8  20.2  77.0  11.8
 8. de-fra03b-ri1-ae12-0.aorta.net     0.0%   134   20.2  22.8  18.8  51.5   4.9
 9. 213.46.177.42                      0.0%   134   30.1  22.4  19.4  41.6   3.8
10. 216.239.59.60                      0.0%   134   20.7  24.8  19.1  49.9   5.6
11. 216.239.57.125                     0.0%   134   20.9  23.3  19.2  42.1   4.7
12. 108.170.232.77                     0.0%   134   23.1  26.2  22.4  42.5   4.5
13. 74.125.37.88                       0.0%   134   53.3  34.5  30.1  58.7   5.2
14. 66.249.95.142                      0.0%   134   33.0  34.7  30.0  56.7   5.4
15. 216.239.41.129                     0.0%   134   41.0  33.3  30.5  48.5   3.1
16. ber01s15-in-f3.1e100.net           0.0%   134   56.6  34.9  30.1  59.3   5.8
```

13 Deploy file system support

If your router contains a USB port, it can – in theory – be used to access various external USB storage media. Sadly, the required packages are not provided out of the box.

This problem can be remedied by deploying the following packages via opkg:

```
kmod-usb-storage required
 kmod-usb-storage-extras
```

```
block-mount
  kmod-scsi-core
```

In addition to that, a kmod-fs-* package containing the drivers for the file system is required. One small gotcha awaits all those who want to access FAT filesystems – the relevant package goes by the name: "kmod-fs-msdos".

14 Learn more

OpenWRT can be used for a variety of topics not discussed here due to space constraints. The OpenWRT project team provides a set of step-by-step recipes at **https://wiki.openwrt.org/doc/howto/start** – if you feel like implementing something, check whether someone else has already walked the trek for you!

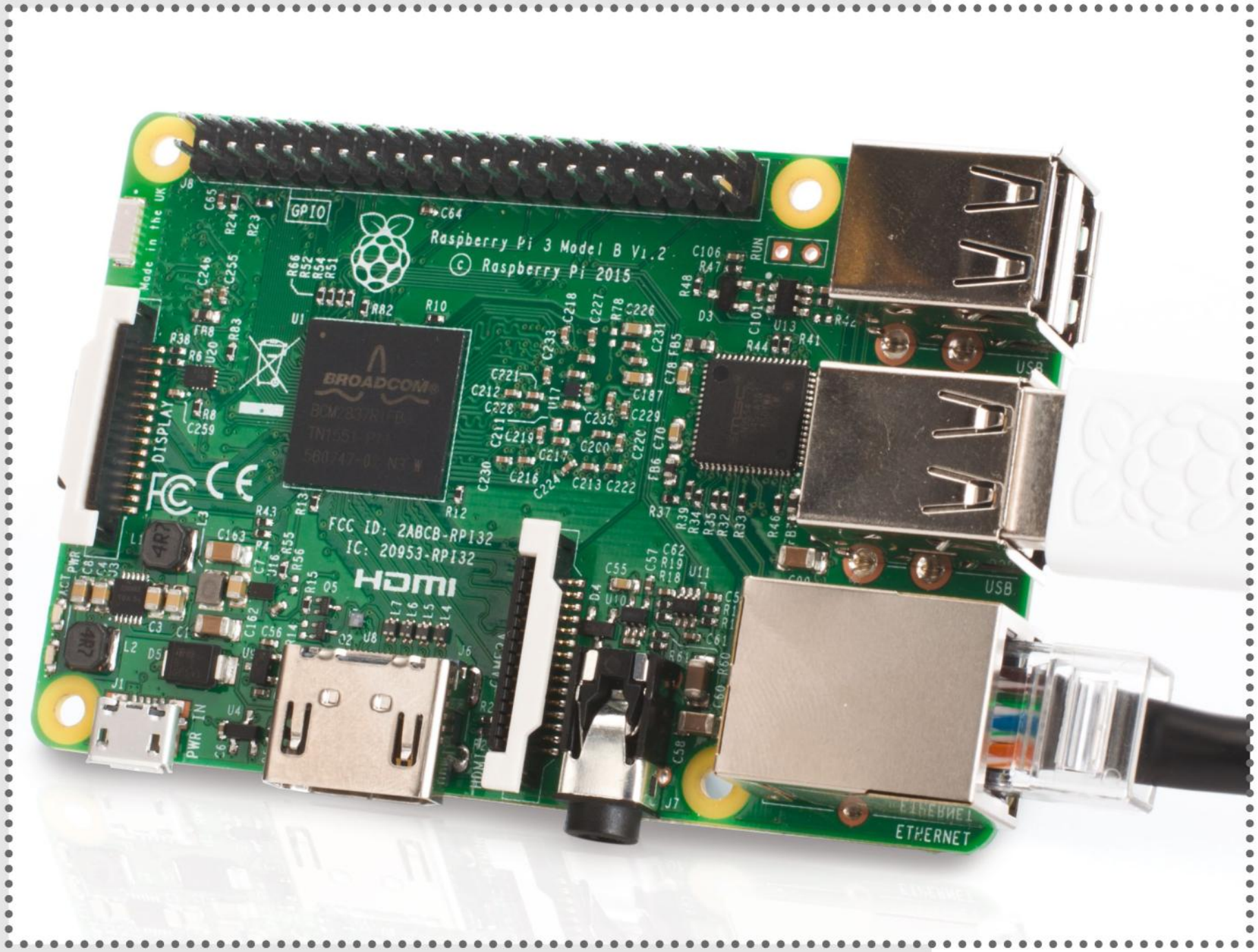
15 Find supported hardware

Our current contraption – made up of a Raspberry Pi and a batch of peripherals – works well for evaluation purposes, but is not particularly well suited to practical deployments. Should you feel like finding a dedicated router, start out by looking at the compatibility list provided at **https://wiki.openwrt.org/toh/start.** Please be aware that router manufacturers tend to change their hardware frequently. In some cases, more than 12 revisions with completely different integrated circuits are known.

16 Hardcore debugging

Should you find yourself locked out of your

OpenWRT router, do not fret. If the memory is not soldered in, then just simply mount it with a cardreader of choice.
Most, if not all, Linux distributions will display the contents of the file systems immediately – accessing some of the files requires that the file manager is run with root rights (sudo nautilus).



Anaconda for the Pi

Python is the language of choice on the Pi, and there are many packages available within the Raspbian repository. Berryconda helps you move out beyond these modules

Python has always been the language of choice for the Raspberry Pi. As a consequence, many Python modules are available as packages within the Raspbian package repository. You can trust that these have been compiled to run on the ARM processor and that they have been tested and are therefore safe to use. But not all Python modules are available this way. For the missing modules, the intention is that you could use pip to install them yourself. While this works fine for some, you will eventually start running into those modules that require debugging to figure out why they aren't working correctly on the ARM processor. The amount of work involved in debugging these types of issues really should be distributed across many people, and luckily it is. For Python coders, a very good distribution of Python modules are available as the Anaconda project. There is a community port called Berryconda, which has been ported to the Raspberry Pi. The main website for the project is github.com/jjhelmus/berryconda. This article will cover the steps to getting it installed and ready to

use on your Raspberry Pi so that we can go on and look at some of the functionality that becomes available to you for Python coding in future articles.

The very first step is to download and install the core of the Berryconda system. On the download page, there are installation files for both armv6l (Raspberry Pi 1 or Zero) and armv7l (Raspberry Pi 2 or 3). There are also different versions for Python 2.X or Python 3.X. The installer is actually a shell script, so all you need to do is to make the script executable and then run it. For example, we can install the Python 3.x version on a Raspberry Pi 1 with the following commands:

```
wget https://github.com/jjhelmus/berryconda/releases/download/v1.0.0/Berryconda3-1.0.0-Linux-armv6l.sh
chmod +x Berryconda3-1.0.0-Linux-armv6l.sh
./Berryconda3-1.0.0-Linux-armv6l.sh
```

By default, this will install the base of an Anaconda environment into the directory berryconda3 in your home directory. You are given the option to change this in case you want to put it elsewhere. The installer also asks whether you want to add the path to the binaries to your PATH environment variable, which is a good idea. Remember to exit from the current shell and log back in so that the new path is picked up. Either that or manually source the initialisation file.

The main utility within Anaconda is the conda packaging system. Conda provides a very fully featured package management system to handle Python modules and all of the various dependencies required.

There is a lot of good documentation available on conda.io/docs, covering all the options and functionality available. The very first thing you will want to do is to keep your current system up to date. You can update individual packages with the command:

```
conda update package-name
```

This command will go to the internet and figure out what new versions of that package exist, and if it finds one, it will ask you whether you really want to update. If you want to just see what will happen, you can use the command line option --dry-run to see what commands would be run. If you just want to keep the entire system updated, you can use this to handle updating everything:

```
conda update --all
```

To install new packages, you need to first find out what has already been packaged and is available. As this is a community effort, the selection will vary over time. So you should check before trying to install some packages. You can do that search with the command:

```
conda search some_text
```

This will do a regular expression search of package titles, looking for the given text, and return a list of any available. Sometimes, however, you may get a rather large list returned. If this happens, you can do a more refined search using the usual regex options used in many other UNIX utilities. If you already know the

package name, you could use the -f option to force the search to return exact matches. If the package is already installed on your Raspberry Pi, it will have an asterisk. When you find and are ready to install the package, use:

```
conda install ipython
```

This will install the latest version of the ipython package in the Berryconda environment. It will also install any missing dependencies, as well as updating any out-of-date ones. While the online documentation is great, there are also help pages within conda. If you want general help, you can get it with the help command. For help with some specific conda commands, you can use something like the following:

```
conda install --help
```

This is handy for all of the details that nobody can seem to remember. Over time, you may find that you don't need all the installed packages are needed, which could be an issue on small machines. The first step is to clean up the Berryconda environment. You can remove unused versions of packages, cached installation tarballs and index caches. Assuming that you want as lean a system as possible, you can clean up the entire environment with the following:

```
conda clean --all
```

If there are installed packages that are no longer needed, you can remove them with the uninstall

"The main utility in Anaconda is the conda packaging system"

command. For example, the following would remove the scipy Python module:

```
conda uninstall scipy
```

If you have a drastic need to restart, you could remove everything from the environment by using the --all command option.

One of the great strengths of Python is the large set of third-party modules available for extending functionality. Unfortunately, this is also one of its weaknesses, leading to a large amount building up over time. The Python module python-env was written to try to deal with this issue. The conda packaging system also understands creating isolated Python environments. This way, you can have the best of both worlds: easier Python module management combined with easier Python environment management. You can get a list of all of the environments conda knows about with:

```
conda env list
```

If you have just installed Berryconda, the only environment listed is the root environment. Now, let's say that you needed to start a new code base, developing the software for a big new project. You could create a new empty environment with the following command:

```
conda create --name deathstar
```

This command creates a new subdirectory within the envs subdirectory inside the Berryconda environment.

If you rerun the environment list command, you will now see both the root environment and the deathstar environment, with the latter tagged by an asterisk as the currently active one. You can now install only the modules you require for this specific software project by activating it and running the usual conda commands. For example, the following would install ipython, along with all of its dependencies, within the deathstar environment.

```
source activate deathstar
conda install ipython
```

You can leave a given environment by running the deactivate script to reset all of the environment variables. If you have already put together a basic environment to use as a starting point for a new environment, you can clone it as your boilerplate. The clone option to the create command comes in handy for such a task.

```
conda create --name deathstar2 --clone deathstar
```

You can always clean up old environments with the remove command in conda.

```
conda --remove deathstar --all
```

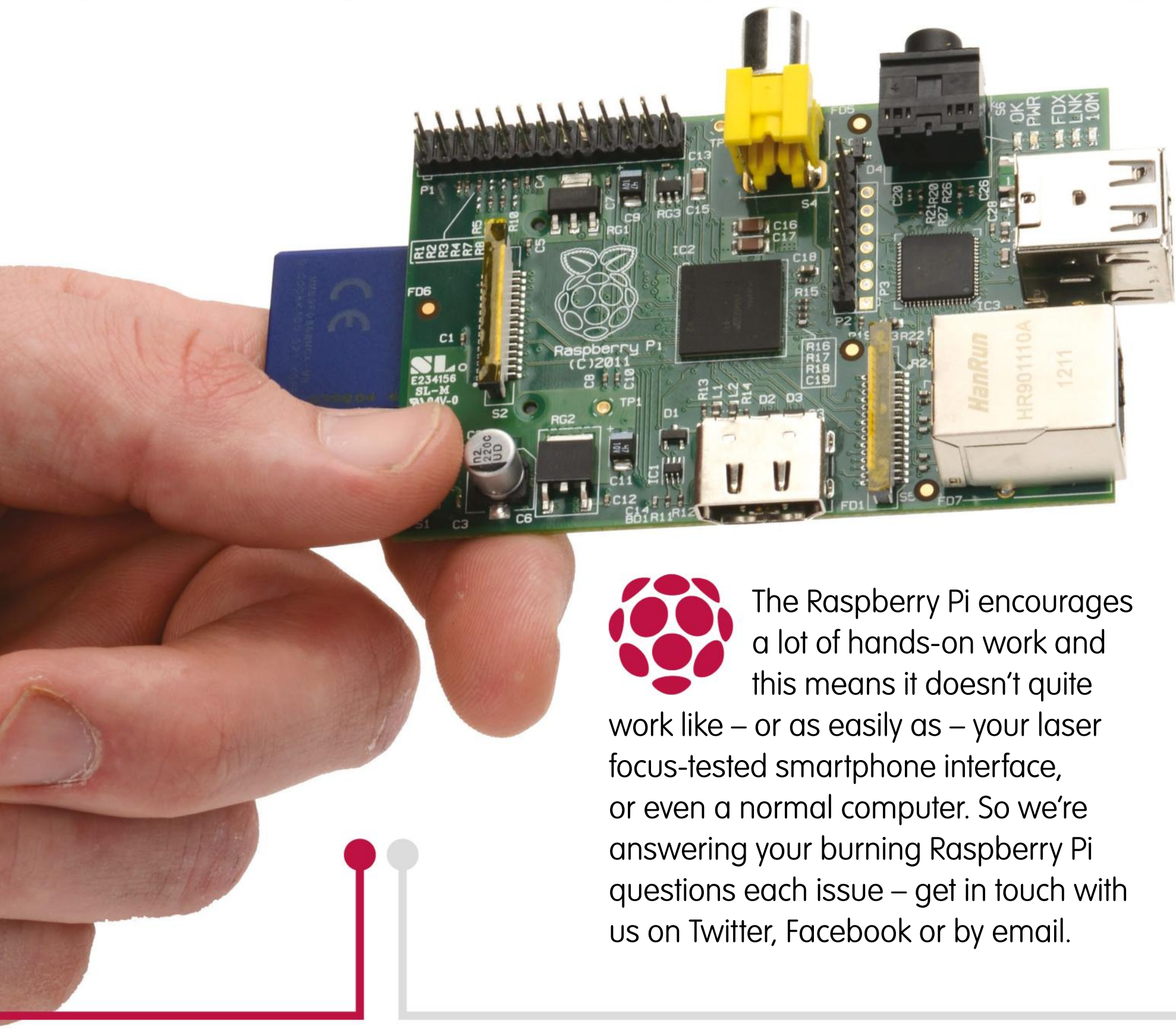
The command option --all ensures that all portions of the environment are deleted during the removal process.

Now you should have all of the ito be able to work in isolated environments for your Pi projects. This is especially useful in the development stages of a new project as you can get the all modules you need.

Talking Pi

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk

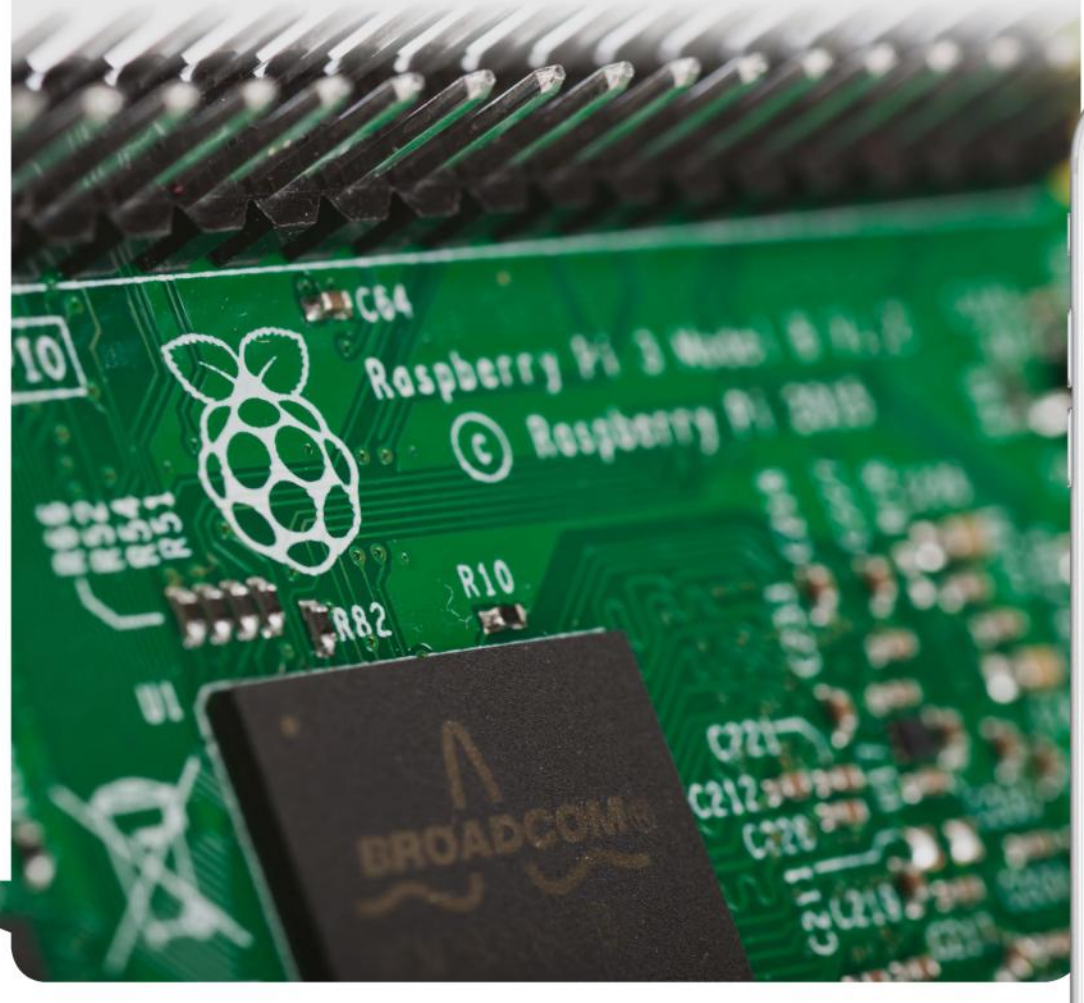


The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

I'm looking to expand my Pi, can I add more memory or swap it out for something a bit bigger?
Tom via email

We're sad to say that upgrading the memory on your Pi isn't something that you'll be able to do, Tom. The RAM changes on the different models of the Raspberry Pi but it's pretty much stuck at what you're given.

So if you've got either a Model A, A+, B, B+ or a Raspberry Pi Zeo it looks like you're going to have to deal with what that model comes with. This is because these models are a package on package that is on top of the SoC, which means that changing or expanding the RAM isn't possible. If you've got a 2B or 3B model, the RAM is actually a seperate chip but the maximum amount that the 2B can support is only 1GB.



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

I'm looking to link up my Raspi Zero with my TV, can I do this?
Elliott via email

You certainly can Elliott! The Raspberry Pi Zero comes with a mini HDMI port so that you can link it up with your TV. In fact, all models of the Raspi have some form of HDMI cable so you're pretty much set for TV viewing.

Most Pis also have a composite so you can hook it up to an old analogue TV through the composite or through a composite to scart connector, to a digital TV or to a DVI monitor

Happy viewing!

Does the Raspi have built in Wi-Fi or do I need to use a dongle?
Lisa via email

Hi Lisa! This depends entirely on which model of the Pi you're using. It's your lucky day if you're looking at a Raspi 3, as it's the only model that actually comes with built in Wi-Fi.

There's nothing to fear, however, as every other model of the Raspberry Pi is capable of supporting and running a Wi-Fi dongle.

To make things even better, The Foundation has its own Wi-Fi dongle that's been fully tested and works perfectly. You can pick that up from their distributors for around £30, but of course, you can always use one from another provider if you want.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Master Pi's Pixel desktop

Get this issue's source code at:

www.linuxuser.co.uk/raspicode